On the Time-Based Conclusion Stability of Software Defect Prediction Models

Abdul Ali Bangash · Hareem Sahar · Abram Hindle · Karim Ali ·

Received: date / Accepted: date

Abstract Researchers in empirical software engineering often make claims based on observable data such as defect reports. Unfortunately, in many cases, these claims are generalized beyond the data sets that have been evaluated. Will the researcher’s conclusions hold a year from now for the same software projects? Perhaps not. Recent studies show that in the area of Software Analytics, conclusions over different data sets are usually inconsistent. In this article, we empirically investigate whether conclusions in the area of defect prediction truly exhibit stability throughout time or not. Our investigation applies a time-aware evaluation approach where models are trained only on the past, and evaluations are executed only on the future. Through this time-aware evaluation, we show that depending on which time period we evaluate defect predictors, their performance, in terms of F-Score, the area under the curve (AUC), and Mathews Correlation Coefficient (MCC), varies and their results are not consistent. The next release of a product, which is significantly different from its prior release, may drastically change defect prediction performance. Therefore, without knowing about the conclusion stability, empirical software engineering researchers should limit their claims of performance within the contexts of evaluation, because broad claims about defect
prediction performance might be contradicted by the next upcoming release of a product under analysis.

Keywords Conclusion Stability, Defect Prediction, Time-aware Evaluation

1 Introduction

Defect prediction models are trained for predicting future software bugs using historical defect data available in software archives and relating it to predictors such as structural metrics (Chidamber and Kemerer, 1994; Martin, 1994; Tang et al., 1999), change entropy metrics (Hassan, 2009), or process metrics (Mockus and Weiss, 2000). The accuracy of defect prediction models is estimated using defect data from a specific time period in the evolution of software, but the models do not necessarily generalize across other time periods.

Conclusion stability is the property that a conclusion remains stable as contexts, such as time of evaluation, change. For example, if the conclusion of a current evaluation of a model on a software product is the same as that of an evaluation done a year ago, then we consider that conclusion to be stable. A lack of conclusion stability would be if the model’s performance is inconsistent with itself across time. Instead of over generalizing our conclusions beyond the period of evaluation, if we claimed the model’s performance was within the period of evaluation, our claim would still hold.

Prior work (Lessmann et al., 2008; Menzies et al., 2010; Turhan, 2012) examined various factors affecting the conclusion stability of defect prediction models. However, none explored the conclusion stability across time. The goal of this paper is to investigate conclusion stability of software defect prediction models and understand how their performance estimates, measured using F-Score, Area under the Curve (AUC), and Matthews Correlation Coefficient (MCC), vary across different time periods. In our evaluation, we carefully consider the time-ordering of versions and ensure our models do not involve time-travel. Time-travel is a colloquial term to describe models that should be time sensitive but are trained on future knowledge that should not be known for predicting defects in the past.

Existing defect prediction studies fail to avoid time-travel because of the choice of a cross-validation evaluation methodology, which randomly splits data into partitions and uses these partitions for training and testing, irrespective of the chronological order of data. As a result, defect prediction models often get trained on future data which is not available, in reality, at the time of training. For example, due to cross-validation, a version released in 2010 may be used for training a model that predicts defects for a version released in 2009. This situation is explained in Table 1 that shows a cross-validation evaluation for three software versions \(v_n\) released between 2008 and 2010. The table shows that not all Training (Tr) and Test combinations are realistic for building defect prediction models, as some will lead to models that are time insensitive (trained on future data). For instance, a case where Tr set = \{v2\} and Test set = \{v1\}, the evaluation seemingly have engaged in time-travel. Rakha et al. (2018) refer to such evaluation as classical evaluation, whereas Hindle and Omiecinski (2019) call it time-agnostic. Many claim that ignoring time provides highly unrealistic performance estimates (Tan et al., 2015; Rakha et al., 2018).
Table 1 An example illustrating three cross-validation settings (a=1/1, b=2/1, c=1/2) of three releases of a hypothetical product over a period of three years (v1=2008, v2=2009, v3=2010).

| Cross Validation | Training/Test | Time-travel |
|------------------|--------------|-------------|
| Training set     | Test set     |             |
| {v1}             | {v2}         | X           |
| {v1}             | {v3}         | X           |
| {v2}             | {v1}         | ✓           |
| {v3}             | {v2}         | X           |
| {v3}             | {v1}         | ✓           |
| {v3}             | {v2}         | ✓           |

(a) Cross-validation (1/1) having 1 release in Training set and 1 release in Test set

| Cross Validation | Training/Test | Time-travel |
|------------------|--------------|-------------|
| Tr set           | Test set     |             |
| {v1,v2}          | {v3}         | X           |
| {v2,v3}          | {v1}         | ✓           |
| {v3,v1}          | {v2}         | ✓           |

(b) Cross-validation (2/1) having 2 releases in Training set and 1 release in Test set

| Cross Validation | Training/Test | Time-travel |
|------------------|--------------|-------------|
| Tr set           | Test set     |             |
| {v1}             | {v2,v3}      | ✓           |
| {v2}             | {v1,v3}      | ✓           |
| {v3}             | {v1,v2}      | ✓           |

(c) Cross-validation (1/2) having 1 release in Training set and 2 releases in Test set

Hindle and Onuczko (2019), yet, there are several just-in-time based approaches that only consider release order for within project defect prediction (Huang et al., 2017; Yang et al., 2016), but engage in time-travel in cross project defect prediction settings (Yang et al., 2016; Kamei et al., 2016; Yang et al., 2015).

In this paper, we evaluate five defect prediction approaches using the publicly available Jureczko dataset (Jureczko and Madeyski, 2010), and show that data from different time periods leads to varying conclusions. In our evaluation, we strictly consider the chronological order of data and propose four generic time-aware configurations that can be used to split the data set into training and testing. The purpose of proposing these configurations is to make the experiment performance-wise scalable for evaluating other approaches in which running all possible Tr-Test set combinations is expensive, such as duplicate bug reports retrieval involving extensive string matching (Hindle and Omuczko 2019).

Our results indicate that the evaluated cross-project defect prediction approaches have limited stability in their conclusions and time-travel produces false estimates of performance. Therefore, while conducting defect prediction studies, researchers should not engage in time-travel and also avoid over generalizing their
conclusions, but instead couch the claims of performance within the contexts of evaluation. To summarize, the main contributions of this paper are:

- A methodology for time-aware evaluation of defect prediction approaches;
- A case study of conclusion stability in cross-project defect prediction with respect to time;
- A comparison of the performance rankings of five cross-project defect prediction approaches using time-aware evaluation with performance of time agnostic evaluation;
- Guidelines for researchers and practitioners for the time-aware evaluation of defect prediction models.

2 Related Work

Software defect prediction has a plethora of approaches with the earliest proposals dating back to the 1990s where linear regression models based on Chidamber and Kemerer (CK) metrics (Chidamber and Kemerer, 1994) were used to determine the fault proneness of classes (Basili et al., 1996). A number of metrics have been used since then as indicators of software quality such as previous defects (Zimmermann et al., 2007), process metrics (Hassan, 2009), Rahman and Devanbu (2013), and churn metrics (Nagappan and Ball, 2005). Within project defect prediction (WPDP) uses data from the same project for training and testing whereas in cross project defect prediction (CPDP), training and testing data comes from different projects. Several approaches for both WPDP (Turhan et al., 2009; Basili et al., 1996) and CPDP (Zimmermann et al., 2009; Peters et al., 2013; Nam et al., 2013) are available in the literature. There have also been benchmark studies on both types of defect prediction (D'Ambros et al., 2012; Herbold et al., 2018). WPDP approaches have better performance while CPDP approaches are likely transferable to other projects with certain limitations (Zhang et al., 2014). Herbold (2017b) conducted a systematic mapping of defect prediction literature with a focus on cross project defect prediction approaches. They identified that the results of studies are not comparable due to the lack of use of common data sets and experimental setups.

In their follow up work, Herbold et al. (2018) replicated 24 defect prediction approaches using 5 publicly available data sets and multiple learners. Their goal was to benchmark the defect prediction approaches using common data sets and metrics so that state-of-the-art approaches can be ranked according to their performance using Area under Curve (AUC), F-Score, G-measure, and Matthews Correlation Coefficient (MCC) metrics. Jureczko (Jureczko and Madeyski, 2010) is one of the well-known defect prediction data sets which was also used in the benchmarking study. It originally contains open-source, proprietary and academic projects but Herbold et al. (2018) used only 62 versions of several open-source and academic projects.

Prior to this paper, conclusion stability has been analyzed by several researchers. Lessmann et al. (2008) and Menzies et al. (2010) investigated the effect of classifiers, trained using same data, on the quality of prediction models whereas Ekanayake et al. (2012, 2009) investigated the effect of data set and concept drift respectively. Lessmann et al. (2008) found statistically significant difference among the performance of two classifiers and Menzies et al. (2011) observed inconsistent
conclusions for different clusters within the same data. Inspired by this prior work, another set of experiments were conducted by D'Ambros et al. (2012) to rank approaches across several data sets following a statistically sound methodology. In more recent work, Tantithamthavorn et al. (2018) concluded that parameter optimization can significantly impact the performance stability, and ranking of defect prediction models. This view is similar to Menzies’ view who argued that a learner tuned to a particular evaluation criterion, performs best for that criterion, hence it shall be critically chosen (Menzies et al., 2010).

Tantithamthavorn et al. (2015) in their work show that issue report mislabelling significantly impacts the defect prediction models. In a later comparison study, Tantithamthavorn et al. (2017) concluded that the choice of model validation technique for defect prediction models can also affect performance results. Tan et al. (2015) identified that cross-validation produces false precision results for change classifications and addressed the problem using time-sensitive and online change classifications. Their emphasis is on removing imbalances in data using re-sampling techniques for better change classifications. Turhan (2012) also studied the conclusion instability caused due to data set shift but their focus was not specific to defect prediction, rather on software engineering prediction models in general. Similarly Krishna and Menzies (2018) show that there can be large differences in conclusions depending on different source data sets and suggest mitigating the problem with the help of bellwethers. Bellwethers seem to restrain instability but based on the results of our study, we consider it of utmost importance to keep regard of time while finding out the bellwether project. However, we believe this work complements our work.

Time-agnostic evaluation has been criticized as unrealistic by Hindle and Omuczek (2019) who argue that the results based on a time-agnostic evaluation might not be applicable to any real-world context. Yang et al. (2016) hold a similar view and motivated by Sliwerski et al. (2005a) they adopted a time-wise cross-validation within projects for evaluating the prediction effectiveness of unsupervised models. However, in their cross-project defect prediction setting, they seem to be time travelling again. Instead of using their approach we propose four time-aware configurations to avoid discarding some of the valid models that time-wise cross-validation will not generate. Jimenez et al. (2019) assessed the impact of disregarding temporal constraints on the performance of vulnerability prediction models and found that the otherwise highly effective and deployable results quickly degrade to an unacceptable level when realistic information is considered. Their work is limited to the prediction of vulnerabilities though, which are just a subset of defects. Rakha et al. (2018) also claim that time-agnostic evaluation overestimates performance. They argue that the range of performance estimates, rather than a single value should be reported.

3 Methodology

In this section, we explain a time-aware evaluation methodology that we follow for building the defect prediction models that do not involve time travel. Researchers can use this methodology for the evaluation of defect prediction techniques in future work to avoid time-agnostic evaluation.
Table 2 An example illustrating four time-aware settings (a=CC, b=IC, c=CI, d=II) of three releases of a hypothetical product over a period of three years (v1=2008, v2=2009, v3=2010). φ = empty set representing no release available at that time. ∞ = max window size possible.

| Tr set  | Test set  | Split point | Window size |
|---------|-----------|-------------|-------------|
| {v1}   | {v2}      | 2008-2009  | 1           |
| {v2}   | {v3}      | 2009-2010  | 1           |
| {φ,v1} | {v2,v3}   | 2008-2009  | 2           |
| {v1,v2} | {v3,φ}   | 2009-2010  | 2           |
| (a) Configuration Constant-Increasing (CC) |

| Tr set  | Test set  | Split point | Window size |
|---------|-----------|-------------|-------------|
| {v1}   | {v2}      | 2008-2009  | 1           |
| {v1,v2} | {v3}     | 2009-2010  | 1           |
| {v1}   | {v2,v3}   | 2008-2009  | 2           |
| {v1,v2} | {v3,φ}   | 2009-2010  | 2           |
| (b) Configuration Increasing-Constant (IC) |

| Tr set  | Test set  | Split point | Window size |
|---------|-----------|-------------|-------------|
| {v1}   | {v2,v3}  | 2008-2009  | 1           |
| {v2}   | {v3}      | 2009-2010  | 1           |
| {φ,v1} | {v2,v3}   | 2008-2009  | 2           |
| {v1,v2} | {v3,φ}   | 2009-2010  | 2           |
| (c) Configuration Constant-Increasing (CI) |

| Tr set  | Test set  | Split point | Window size |
|---------|-----------|-------------|-------------|
| {v1}   | {v2,v3}  | 2008-2009  | ∞           |
| {v1,v2} | {v3}     | 2009-2010  | ∞           |
| (d) Configuration Constant-Increasing (II) |

3.1 Select techniques to evaluate

The first step is to select techniques for validation, and these can either be newly proposed techniques or existing defect prediction proposals. In general, defect prediction techniques can be selected from a broad category of within project defect prediction techniques (WPDP) or cross project defect prediction techniques (CPDP). As the name suggests, WPDP uses the same project in training and testing, whereas CPDP is across different projects. CPDP has several variants including strict CPDP, mixed CPDP, and pair-wise CPDP (Herbold, 2017b). In strict CPDP, there is a strict distinction between the projects used in training and testing. This restriction implies that none of the projects used for training the model remain part of the testing data so that information from same context does not mix up. Contrarily, in mixed CPDP, some releases of a project are used for training while others are used for testing. In pair-wise CPDP, a separate model is trained using each project release, and their performance is averaged for estimating the actual performance.
3.2 Extract software defect prediction metrics with dated releases

Existing software systems with issue trackers can be used to extract software defect prediction metrics and post-release defects via mining software repositories. Extraction methodologies discussed in prior work ([Sliwerski et al., 2005b; Fischer et al. 2003; Zimmermann and Nagappan, 2007]) can be leveraged for the purpose of gathering data. We can alternatively benefit from existing defect data sets used by prior studies for evaluating the technique. One has to make sure that the data set contains releases that have dates or time-stamps. Alternatively, if versions are specified, one can extract and use version release dates. For example, if the data set contains commit history ids, bug report ids, and version release tags, we can extract version release dates from these factors. Before moving on to the next step, one has to label the defect data set instances with dates or timestamps.

3.3 Sort and Split project versions into time buckets

In this step, the defect data set is first sorted according to the time available in the form of version dates, and then split using N split points. A split point is the reference point in time that partitions the defect data into time-buckets, and it is chosen such that the data is partitioned into a day, month, or year granularity. Consequently, each time-bucket can span days, months, or years of releases. If the defect data set contains dates at day level granularity, i.e., day of the release date is available, the split point can be kept to day level or higher. For example, for a data set of 10 years with month of the release dates available, the choice of a monthly split point leads to \(10 \times 12 = 120\) buckets for that data set. However, each bucket may have an uneven distribution of projects or project versions. Figure 1 further illustrates how an example 4-year data set is divided into four buckets using split point at one year granularity. Bucket-1 is formed starting from the oldest project version until the first split, so it contains project versions spanning a year. Bucket-2 contains one year data between first and second split, and so on.

These split points allow the software versions before a certain split to be used for training set while any versions after that split form the test set. Unlike cross-validation there is no time-travelling in such evaluation because the buckets are ordered by time. Notice that a lower granularity spreads the data set well across the timeline and a great number of data points are available for constructing and evaluating the defect prediction models. For the rest of this paper, we will refer to these time ordered buckets as a time-series data set.

3.4 Generate Training-Test pairs from time buckets

In this step, we use the time-series data set to generate multiple Training-Test (Tr-Test) pairs following four time-aware configurations. Figure 1 provides a high-level overview of these configurations where the time granularity of buckets is one year, and each bucket contains multiple project versions. In each configuration, the split point divides the data into two parts: past and future. The red dot represents a split point in Figure 1. The buckets containing project versions before the split point form the past of a data set and will be considered for training while those
|         | Year-1 (Y1) | Year-2 (Y2) | Year-3 (Y3) | Year-4 (Y4) |
|---------|-------------|-------------|-------------|-------------|
| Bucket-1 (B1) | P₁V₁ | P₁V₁ | P₁V₁ | P₁V₁ |
| Bucket-2 (B2) | P₂V₁ | P₂V₂ | P₂V₂ | P₂V₂ |
| Bucket-3 (B3) | P₁V₂ | P₁V₂ | P₁V₂ | P₁V₂ |
| Bucket-4 (B4) | P₂V₃ | P₂V₃ | P₂V₃ | P₂V₃ |

**CC Configuration**

- **K=1**
  - \(\text{Tr} = \{B1\}, \text{Test} = \{B2\}\)
  - \(\text{Tr} = \{B2\}, \text{Test} = \{B3\}\)
  - \(\text{Tr} = \{B1,B2\}, \text{Test} = \{B4\}\)

- **K=2**
  - \(\text{Tr} = \{B1\}, \text{Test} = \{B2,B3\}\)
  - \(\text{Tr} = \{B2\}, \text{Test} = \{B3,B4\}\)
  - \(\text{Tr} = \{B1,B2\}, \text{Test} = \{B4\}\)

- **K=3**
  - \(\text{Tr} = \{B1\}, \text{Test} = \{B2,B3,B4\}\)
  - \(\text{Tr} = \{B2\}, \text{Test} = \{B3,B4\}\)
  - \(\text{Tr} = \{B1,B2,B3\}, \text{Test} = \{B4\}\)

**IC Configuration**

- **K=1**
  - \(\text{Tr} = \{B1\}, \text{Test} = \{B2\}\)
  - \(\text{Tr} = \{B2\}, \text{Test} = \{B3\}\)
  - \(\text{Tr} = \{B1,B2\}, \text{Test} = \{B4\}\)

- **K=2**
  - \(\text{Tr} = \{B1\}, \text{Test} = \{B2,B3\}\)
  - \(\text{Tr} = \{B2\}, \text{Test} = \{B3,B4\}\)
  - \(\text{Tr} = \{B1,B2\}, \text{Test} = \{B4\}\)

- **K=3**
  - \(\text{Tr} = \{B1\}, \text{Test} = \{B2,B3,B4\}\)
  - \(\text{Tr} = \{B2\}, \text{Test} = \{B3,B4\}\)
  - \(\text{Tr} = \{B1,B2,B3\}, \text{Test} = \{B4\}\)

**CI Configuration**

- **K=1**
  - \(\text{Tr} = \{B1\}, \text{Test} = \{B2,B3,B4\}\)
  - \(\text{Tr} = \{B2\}, \text{Test} = \{B3,B4\}\)
  - \(\text{Tr} = \{B1,B2\}, \text{Test} = \{B4\}\)

- **K=2**
  - \(\text{Tr} = \{B1\}, \text{Test} = \{B2,B3,B4\}\)
  - \(\text{Tr} = \{B2\}, \text{Test} = \{B3,B4\}\)
  - \(\text{Tr} = \{B1,B2,B3\}, \text{Test} = \{B4\}\)

- **K=3**
  - \(\text{Tr} = \{B1\}, \text{Test} = \{B2,B3,B4\}\)
  - \(\text{Tr} = \{B2\}, \text{Test} = \{B3,B4\}\)
  - \(\text{Tr} = \{B1,B2,B3\}, \text{Test} = \{B4\}\)

**II Configuration**

- **K=∞**
  - \(\text{Tr} = \{B1,B2\}, \text{Test} = \{B3,B4\}\)

**Fig. 1** Generating Training (\(\text{Tr}\)) and Test (\(\text{Test}\)) pairs using four time-aware configurations: Constant-Constant (CC), Increasing-Constant (IC), Constant-Increasing (CI), Increasing-Increasing (II). Pn refers to Project number, Vn refers to Version number, Yn refers to (Year number), and K is Window size and decides the number of time buckets that are used in training and testing. \(\emptyset\) in II means that Window size does not matter in that configuration.
On the Time-Based Conclusion Stability of Software Defect Prediction Models

after the split point form the future and are used for testing (Test). We further employ window size to select the number of time-buckets to be used for generating Tr-Test pairs. The window size also has a granularity in terms of the number of time buckets, e.g., a window size of one corresponds to one year of data in our example. Consequently, the Tr-Test set size, i.e., the number of project versions in training and test set, varies as window size changes: number of project versions is not constant in every bucket.

To explain the four configurations, we use the example introduced earlier in Section 1 and present Tr-Test pairs corresponding to the four time-aware configurations in Table 2. φ in the table represents an empty set for the cases when window size exceeds the number of buckets available in the data set for Tr or Test set.

Configuration 1 — Constant-Constant (CC): In this configuration, the Tr and Test set are populated according to the window size. At each split point with a constant window size $K$, we take $K$ time-buckets before the split point for Tr set and an equal number of buckets after the split point for Test set, as shown in Figure 1. This Tr set and Test set forms a Tr-Test pair. The window size is increased once the Tr-Test pairs over all split points are generated. As a result, we get one Tr-Test pair corresponding to each value of window size and split point.

The process of generating Tr-Test pairs is repeated until all possible pairs corresponding to each split point and window size are generated. There can be cases where an equal number of buckets before and after the split point are not available, for example, if we consider $K=3$ at split Y3-Y4 in Figure 1 there is only one bucket left for testing. To ensure consistency in generating configurations, we consider as many buckets as available at such split points, hence our Test set = \{B4, φ, φ\}. This configuration is similar to the evaluation of Rakha et al. 2018 except that they employed tuning.

Configuration 2 — Increasing-Constant (IC): At each split point in this configuration, the Test set is populated with $K$ time buckets after the split point, where $K$ is the window size. While the Tr set is populated with all the time-buckets available before the split point. Same as CC, the window size is increased once the Tr-Test pairs over all split-points are generated. Considering each split point and current window size value referred to as $K$ in Figure 1, we take all time-buckets before the split point for Tr and $K$ number of buckets after the split point for Test. The example Tr-Test pairs corresponding to each value of window size and split point are shown in Figure 1.

Configuration 3 — Constant-Increasing (CI): Contrary to IC, at each split point in this configuration, the Tr set instead of Test set is populated with $K$ time buckets before the split point, where $K$ is the window size. Whereas same as CC and IC, the window size is increased once the Tr-Test pairs over all split-points are generated. Considering each split point and current window size value referred $K$; we take $K$ number of buckets before the split point for Tr while all time-buckets after the split point for Test. The example Tr-Test pairs corresponding to each value of window size and split point are shown in Figure 1.

Configuration 4 — Increasing-Increasing (II): In II, the window size does not matter because at each split point, the Tr-Test pairs are generated by taking all the buckets before split point for training and all those after the split point for testing. We set window size or $K$ in this configuration to infinity as that is theoretically the maximum possible window size.
Each of the four configurations serves a different purpose, and depending on the context one configuration is a more appropriate choice than the other. For example, the quality assurance team wanting to test the next due release of a project against the entire past may use IC or II configurations. The CI configuration is more useful in cases where a major release in the past has entirely changed the system, and the developers want to test their system since then. CC and II configurations might benefit researchers who are trying to evaluate the defect prediction methodologies, so they can evaluate and compare the performance of defect prediction approaches.

3.5 Build prediction models and evaluate performance

Each technique applies certain treatment on the instances in the training and test set before building the model. For example, one technique may apply log transformation on the training set, while another may use K-Nearest Neighbours (KNN) relevancy filtering. Therefore, we apply the treatment proposed by a defect prediction technique to all the Tr and/or Test sets generated in the previous step and then build a prediction model from each Tr set. We evaluate the model on Test set and collect the performance measures. Finally, we have performance metrics corresponding to evaluation of approach at each time split and for a specific window size of each configuration.

4 Experimental Setup

In this section, we investigate the conclusion stability of defect prediction approaches by employing the proposed configuration settings.

4.1 Select techniques to evaluate

In this work, we do not propose a new defect prediction approach. Instead, we re-evaluate existing defect prediction techniques from the literature. Specifically, we evaluate the conclusion stability of five defect prediction techniques that Herbold et al. (2018) recently evaluated in a defect prediction benchmarking study. We choose this study as a reference, because it is the most comprehensive evaluation of CPDP approaches, and evaluating techniques from their study allows us to compare our results with them. The results and replication kit of benchmarking study are also publicly available (Herbold 2017a).

The five replicated techniques include the one proposed by Amasaki et al. (2015) (Amasaki15), Watanabe et al. (2008) (Watanabe08), Cruz and Ochimizu (2009) (CamargoCruz09), Nam and Kim (2015) (Nam15), and Ma et al. (2012) (Ma12). The selection is guided by original rankings reported in the benchmarking study done by Herbold et al. (2018). CamargoCruz09 and Watanabe08 are the top-ranked techniques according to the rankings reported in Herbold et al. (2018). The other two techniques, Amasaki15 and Ma12 are among the middle ranked
approaches whereas Nam15 performs worst. Hence, to ensure diversity, we choose two top ranked, two middle ranked, and one lowest rank approach for evaluation.

We take a limited number of techniques, because of the large number of models that we already have to train at each point in time with varying window sizes. Our problem has a huge dimensionality and it could grow significantly by adding more techniques, because, for each new technique multiple Tr-Test pairs i.e. models need to be evaluated.

4.2 Extract software defect prediction data set with dated releases

To choose our data set, we explored the well-known PROMISE repository that is used in many defect-prediction studies (Fenton et al., 2007; Menzies and Dr Stefanou 2004; Menzies et al., 2004; Koru and Liu, 2005; Morasca and Ruhe, 2000). Unfortunately, we could not find time-relevant features within that data set, which suggests the lack of concern about the time-order of defect data in the community. We also explored the five data sets used in the benchmarking study of Herbold et al. (2018), but all except the Jureczko (Jureczko and Madeyski, 2010) lack time-relevant information that can be used to retrieve time of occurrence of defects. Since we need release-time information, we only use a subset of Jureczko data set consisting of only open-source projects, and we refer to it as FilterJureczko. We use open-source projects because their version numbers were specified, and hence release dates of only these versions could be retrieved from the project’s version control repositories. As a result, we got 33 versions of 14 open-source projects for our experiment containing 20 static product metrics for Java classes and the number of defects found at class-level.

4.3 Sort and Split project versions into time buckets

The project versions in the FilterJureczko data set are spread across 8.5 years starting from November 1999 and ending at February 2009. We sort the entire data set using the version release dates and then divide it using split points having 6 month granularity. These points equally split the data set into a number of 6 month time-buckets; each containing project versions that are at most 6 months apart. We did not keep a finer granularity than 6 months, because of the limited data at hand and also because project releases are usually several months apart. In total, we have 19 buckets. Each bucket consists of multiple versions of different projects that lie within the 6-month time period. Out of 19 buckets, some buckets have multiple versions of the same project, because multiple versions were released within the 6-month time period whereas some buckets are completely empty because no project version was released during six months. In the end, we partitioned the entire data set into 19 sorted time-buckets and we refer to it as a “time-series data set”. Figure 2 is a graphical illustration of different project versions spread across 19 time buckets. For example, the first bucket has only one

---

1 In the rest of the paper, we do not use the rankings reported in original study of Herbold et al. (2018), but instead use our re-implementation results of his methodology on open-source projects in Jureczko data set.
version of Xerces, and the last bucket has four versions of Camel and one version of Ivy.

4.4 Generate Train-Test pairs from time buckets

We generate multiple Tr-Test pairs from the time-series data set using four generic configurations; CC, IC, CI, and II. The Tr and Test sets are formed through a union of projects data, however, for each pair we make sure that the test set does not contain versions from any project that was part of the training set. We generated a total of 976 Tr-Test pairs for each technique: 318 for CC; 316 for IC; 324 for CI; and 18 for II. As a result, we trained a total of $976 \times 5 = 4880$ models for evaluation of the five techniques that we studied. The different number of Tr-Test pairs (and models) in CC, CI, and IC is due to the strict CPDP settings of our experiment, which does not allow the same project to be used for both training and testing. Consequently, at some split points, there is no data left for testing and hence we eliminate that pair. Figure 3 shows the size of training and test data for each of the pairs in the four configurations. We also show the percentage of defected instances in our training and test data set at each split point and window size in Figure 4.
Fig. 3 Training and Test Data Size ▶KA: what about them? the caption should indicate what it is for. ▶Tr(i) represents training set and Test(i) represents test set at i-th split point in time. K represents window size (irrelevant in II). The color intensity corresponds to number of instances.
Fig. 4 Percentage of Defected Instances in Training and Test data set. Tr(i) represents training set and Test(i) represents test set at i-th split point in time. K represents window size (irrelevant in II). The color intensity corresponds to higher number of defected instances.
4.5 Build prediction models and evaluate performance

The defect prediction techniques apply certain treatments on the data before training the actual model. The treatments are applied as suggested by the benchmarking study of [Herbold et al., 2018]. Suppose the training data is referred as $S$ and the test data is $S^\ast$.

For **Amasaki15** (Amasaki et al., 2015), we perform attribute selection over log transformed data by discarding attributes whose value is not close to any metric value in the data. We then apply relevancy filtering similarly by discarding instances whose value is not close to any instance values.

For **Watanabe08** (Watanabe et al., 2008), we standardize the training data for all Tr-Test pairs as:

$$\hat{m}_i(s^\ast) = (m_i(s^\ast) \cdot \text{mean}(m_i(S))) / (\text{mean}(m_i(S^\ast)))$$

For **CamargoCruz09** (Cruz and Ochimizu, 2009), we use Test data as reference point and apply logarithmic transformation as:

$$\hat{m}_i(s) = \log(1 + m_i(s)) + \text{median}(\log(1 + m_i(S))) - \text{median}(\log(1 + m_i(S^\ast)))$$

For **Nam15** (Nam and Kim, 2015), clustering and labelling of instances is performed based on the metric data by counting the number of attribute values that are above the median for that attribute. Afterwards all instances that do not violate a metric value based on a threshold called metric violation score are selected.

For **Ma12** (Ma et al., 2012), weighting is applied on data on the basis of similarity. The weights are calculated as:

$$w_s = \frac{\text{simatts}_s}{(p - \text{simatts}_s + 1)}^2$$

where $p$ is the number of attributes and $\text{simatts}$ are those attributes of an instance whose value is within the range of test data.

More details about these techniques are available in their original publications. The source code for applying these treatments is provided by [Herbold, 2015, 2017a] as a replication package.

For each technique, we built 976 separate defect prediction models utilizing all the Tr-Test pairs. We trained these models on Decision Trees (DT) using the J48() algorithm in Weka (Witten et al., 2016). We chose DT, because all the studied techniques performed best on Decision Trees classifier in the benchmarking study [Herbold et al., 2018]. To compare our results with the benchmarking study, we also trained our models on DT using a confidence interval of 0.30 with pruning. We did not tune our classifier to keep the experimental settings consistent with [Herbold et al., 2018], because changing them could bias our results. Moreover, our small data set limits us from giving up a whole window for tuning. [Rakha et al., 2018] had an ample amount of data, hence they tuned their models in the duplicate issue reports study.

[2] https://crosspare.informatik.uni-goettingen.de/
While evaluating our models, we calculated their performance in terms of precision, recall, F-Score, MCC, and AUC. Recall is the ratio of true positives to true positives and false negatives, and it measures the number of actual defects that are found. Precision is the ratio of true positives to true positives and false positives, and it measures how many of the found defects are actually defects. F-Score is a combination of precision and recall, and is calculated using the harmonic mean of the two. Matthews Correlation Coefficient (MCC) measures the correlation between the actual and the predicted classifications, ranging between -1 and +1, where -1 indicates total disagreement, +1 indicates perfect agreement, and 0 indicates no correlation at all. AUC is the Area under the Receiver Operating Characteristic Curve, which is a plot of the true positive rate vs the true negative rate.

5 Results

As a result of running our time-aware experiment we gather models for each Tr-Test pair representing one split point in time and a particular window size of a given configuration. All the models are built using Decision Tree classifier and the results constitute a range of performance estimates that we use to examine conclusion stability of defect prediction models across different evaluations. We also compare the results of our time-aware experiment with results obtained by re-conducting the experiment of [Herbold et al. (2018)] only on the FilterJureczko, because time information could only be retrieved for those projects. Hence, instead of reporting the result of Herbold’s original study, we use our re-implementation results of his methodology referred subsequently as HerboldMethod.

5.1 RQ1: Are the defect prediction approaches stable in terms of their conclusions when evaluated over different configurations of time?

Motivation. Prior research evaluates defect prediction approaches in a time-agnostic manner. The results obtained from one specific evaluation at a particular point in time are generalized to all available time-periods. This assumption is unrealistic as defect prediction approaches might not have stable conclusions and hence results cannot be generalized across the entire data set irrespective of time. The goal of this research question is to study the conclusion stability of defect prediction approaches. We hypothesize that “a defect prediction technique has stable conclusion if standard deviation of its F-score performance produced by all Tr-Test pairs in a specific configuration is less than 0.1”. Prior works such as [Zhang et al. (2014)] and [Herbold et al. (2018)] consider 2% and 5% respectively to be a significant performance gain in terms of AUC or F-Score and 0.1 is pretty conservative in reference to these thresholds.

Result. We evaluate five defect prediction approaches in this paper and according to the results of our experiment these approaches have unstable conclusions. To investigate the conclusion stability; we analyze the F-Score values obtained from different evaluations of the five approaches using Tr-Test pairs generated according to the four configurations introduced earlier. The mean F-Score values and
Fig. 5 Comparison of F-Scores of techniques when evaluated over four configurations. A-Amasaki15, B-Watanabe08, C-CamargoCruz09, D-Nam15, E-Ma12. Horizontal line shows HerboldMethod F-Score.
Table 3 Arithmetic Mean and Standard Deviation (SD) of F-Scores of five evaluated approaches using four time-aware configurations.

| Configuration | Amasaki15 Mean | SD | Watanabe08 Mean | SD | CamargoCruz09 Mean | SD | Nam15 Mean | SD | Ma12 Mean | SD |
|---------------|---------------|----|-----------------|----|-------------------|----|------------|----|-----------|----|
| CC            | 0.509         | 0.129 | 0.513           | 0.104 | 0.502             | 0.120 | 0.587      | 0.080 | 0.513     | 0.127 |
| IC            | 0.509         | 0.133 | 0.514           | 0.103 | 0.502             | 0.112 | 0.583      | 0.0714 | 0.508    | 0.131 |
| CI            | 0.511         | 0.127 | 0.515           | 0.101 | 0.499             | 0.117 | 0.590      | 0.076 | 0.511     | 0.125 |
| II            | 0.510         | 0.139 | 0.517           | 0.107 | 0.498             | 0.119 | 0.588      | 0.059 | 0.507     | 0.137 |

The bold values in Table 3 indicate that the overall standard deviation of F-Scores observed across different evaluations in a configuration is greater than 0.1. This is the case for four out of five approaches we evaluated implying that their F-Scores vary by more than 0.1 for several Tr-Test pair evaluations. It confirms our hypothesis that conclusions change depending on the context, i.e., time at which model was trained and evaluated.

Figure 5 shows F-Scores plotted on y-axis over split points in time on the x-axis. The boxplots in figure illustrate the variance in the F-Score values of techniques evaluated according to four configurations. The length of barplots signify the magnitude of variation in the F-Score at a particular split point. If we observe the F-Score values along the timeline in Figure 5, there is a drastic variation at different points in time, particularly for CC and IC and also in CI, but to a relatively lesser extent. In the II configuration the F-Scores of all techniques except Nam15 exhibit a similar variation across timeline. Overall Amasaki shows the highest deviation by deviating more than 0.1 from its mean value almost 71.41% of the times followed by Ma12, CamargoCruz09, Watanabe08 and Nam15 respectively which deviate 68.9%, 45.5%, 42.2% and 12.4% of the times respectively.

Since the time-agnostic evaluation ignores time, therefore all prior works report aggregate F-Score over the entire evaluated time-period. The green constant horizontal line drawn over Figure 5 refers to the F-Score value obtained by HERBOLDMETHOD. The large number of results falling on both sides of the horizontal line indicate that conclusions drawn about the performance of an approach are not stable over different evaluations. For example, at split point 16 in CC configuration in Figure 5-D, F-Score is around 0.8 but it drops around 0.2 if we move just one split point ahead on the timeline. Such cases show how performance claims can be highly unrealistic if the context is ignored. Therefore reporting a single value and generalizing it over different points in a project’s evolution is quite misleading.

The problem is further aggravated by large number of outliers that can be seen in Figure 5, indicating the fact that evaluation can often yield very high or low performance estimates, which are far from the real performance that a defect prediction technique may achieve in practice. Therefore, the conclusions drawn from a specific period of time should not be generalized outside of it. It is rather more appropriate for researchers to report a range of values of a performance metric corresponding to multiple time-periods and contexts of evaluation.
Table 4 Results of Wilcoxon rank-sum tests and Cliff’s Delta tests for F-Score based comparison between four configurations and HerboldMethod

| Configuration | p-value for F-Score | Cliff’s Delta Effect Size |
|---------------|---------------------|---------------------------|
| CC            | < 0.01              | Medium                    |
| IC            | < 0.01              | Medium                    |
| CI            | < 0.01              | Medium                    |
| II            | < 0.01              | Medium                    |

The defect prediction techniques do not have stable conclusions when evaluated over several different points in time using four configurations. All the techniques except Nam15 have F-Scores that deviate more than 0.1 from their mean values in all configurations, and Nam15 deviates the least. This deviation signifies that the performance based on one evaluated period of time cannot be generalized across the entire project or data set irrespective of time. Researchers should carefully couch the results of defect prediction studies against the time-periods of evaluation.

5.2 RQ2: How do the results of time-agnostic and time-aware evaluations differ?

Motivation. The time-agnostic evaluation of defect prediction techniques might lead to false estimates of performance. In this question we compare the results of time-agnostic and time-aware evaluations to better understand the impact of evaluation method on the results of defect prediction models.

Result. We use Wilcoxon rank-sum tests to estimate whether the differences between HerboldMethod and our results are statistically significant or not. Table 4 reports the p-values of Wilcoxon tests and bold values indicate statistically significant differences. The value of $\alpha$ for these tests was set to 0.01 instead of commonly used 0.05 threshold because we want to account for multiple hypothesis testing through Bonferroni correction. The comparison reveals that all the four configurations, CC, IC, CI, and II differ from HerboldMethod in terms of F-Score metrics and the difference is statistically significant (p-value < 0.01).

To quantify the differences between our configurations and HerboldMethod we employ Cliffs Delta which is a measure of the effect size and does not assume normality of distribution. For Cliffs Delta we use the interpretations of Romano et al. (2006) which considers difference to be Negligible if Cliffs $|d| \leq 0.147$, Small if Cliffs $|d| \leq 0.33$, Medium when Cliffs $|d| \leq 0.474$, and Large otherwise. For F-Score, the effect size test results reported Table 4 show that the differences for all configurations have medium effect sizes. These differences suggest that the performance is different for our time-aware and Herbold’s time-agnostic evaluations. Therefore time-agnostic evaluation should not be used to generalize the performance of defect prediction techniques.
The Wilcoxon rank-sum tests suggest that there is a statistically significant difference between the performance of four configurations and HerboldMethod. Also, Cliff’s Delta effect sizes show that these differences are medium for the F-Score performance metrics. These results imply that the traditional evaluation might produce too high or too low performance estimates.

5.3 RQ3: What is the ranking of evaluated techniques in time-aware experiment?

Motivation. The recent replication done by Herbold et al. (2018) ranks 24 cross project defect prediction approaches using common data sets and performance metrics. The aim of their work was to benchmark the performance of CPDP approaches using multiple learners and data sets. We on the other hand claim and show that their conclusion might not hold under different contexts of evaluation. In this research question, we investigate if the rankings of HerboldMethod still hold under our experimental settings or not.

Result. The performance estimates of our four configurations in comparison with HerboldMethod are reported in Table 5. The prior analysis in RQ2 suggests that performance in terms of F-Score differs. We further rank the defect prediction techniques on the basis of all performance metrics. Following the methodology of Herbold et al. (2018) we calculate the Mean Rank Score of a technique per configuration using F-Score, MCC, and AUC metric. Incorporating all the performance metrics into the ranking of techniques eliminates the bias that could arise due to a single metric which fails to estimate model performance.

Our results shown in Table 6 suggest that the ranks of four techniques vary in comparison with HerboldMethod for each configuration. However, Nam15 outperformed other approaches and obtained the top rank in all configurations which is also consistent with HerboldMethod results. It is the only technique whose ranking remains consistent across the four configurations owing to the least variation across all the Tr-Test pair evaluations.

On the other hand, Figure 6 through Figure 9 illustrate that Amasaki15, Watanabe08, CamargoCruz09, and Ma12 remain inconclusive not just across configurations but also at different split points. Furthermore, despite consistent performance of Nam15 in all four configurations, occasional decline at different split points can be observed.

To quantify the variation in the ranks of techniques, we calculate standard deviation of ranks within each configuration and present it in Table 7. The values of standard deviation range from 0.77 (smallest) in Nam15 to 1.23 (largest) in CamargoCruz09 which shows that the ranks of techniques vary or deviate by at least $\pm 1$ when evaluated at different time splits within a configuration. This variation shows that the performance of each technique varies depending on the context of evaluation and the ranks do not generalize over all time-periods.
Table 5 Results comparing HerboldMethod and time-aware evaluation - HerboldMethod reports only one value of F-Score, AUC and MCC for each technique which is duplicated across all rows.

| Technique       | Configuration | HerboldMethod Values | New Values |
|-----------------|---------------|----------------------|------------|
|                 | F-Score | MCC  | AUC  | F-Score | MCC  | AUC  |
| Amasaki15       | CC      | 0.509 | 0.006 | 0.499 | 0.388 | 0.175 | 0.578 |
|                 | IC      | 0.509 | 0.008 | 0.499 | 0.388 | 0.175 | 0.578 |
|                 | CI      | 0.510 | 0.004 | 0.498 | 0.388 | 0.175 | 0.578 |
|                 | II      | 0.510 | 0.004 | 0.498 | 0.388 | 0.175 | 0.578 |
| Watanabe08      | CC      | 0.513 | 0.002 | 0.502 | 0.392 | 0.109 | 0.563 |
|                 | IC      | 0.514 | 0.004 | 0.501 | 0.392 | 0.109 | 0.563 |
|                 | CI      | 0.515 | 0.001 | 0.499 | 0.392 | 0.109 | 0.563 |
|                 | II      | 0.517 | 0.003 | 0.499 | 0.392 | 0.109 | 0.563 |
| CamargoCruz09   | CC      | 0.502 | 0.010 | 0.506 | 0.389 | -0.086 | 0.468 |
|                 | IC      | 0.502 | 0.001 | 0.499 | 0.389 | -0.086 | 0.468 |
|                 | CI      | 0.499 | 0.004 | 0.500 | 0.389 | -0.086 | 0.468 |
|                 | II      | 0.498 | -0.002 | 0.497 | 0.389 | -0.086 | 0.468 |
| Nam15           | CC      | 0.587 | 0.178 | 0.593 | 0.492 | 0.235 | 0.641 |
|                 | IC      | 0.583 | 0.183 | 0.596 | 0.492 | 0.235 | 0.641 |
|                 | CI      | 0.590 | 0.171 | 0.592 | 0.492 | 0.235 | 0.641 |
|                 | II      | 0.588 | 0.176 | 0.594 | 0.492 | 0.235 | 0.641 |
| Ma12            | CC      | 0.513 | 0.007 | 0.503 | 0.392 | 0.160 | 0.581 |
|                 | IC      | 0.508 | 0.005 | 0.503 | 0.392 | 0.160 | 0.581 |
|                 | CI      | 0.511 | 0.001 | 0.498 | 0.392 | 0.160 | 0.581 |
|                 | II      | 0.507 | -0.0001 | 0.499 | 0.392 | 0.160 | 0.581 |

Table 6 New ranks of techniques based on Mean Rank Score and their comparison with HerboldMethod ranks.

| Technique      | HerboldMethod Ranks | New Ranks |
|----------------|----------------------|-----------|
|                | CC | IC | CI | II |
| Nam15          | 1  | 1  | 1  | 1  |
| Ma12           | 2  | 2  | 3  | 4  |
| Amasaki15      | 3  | 4  | 3  | 2  |
| Watanabe08     | 4  | 3  | 2  | 3  |
| CarmagoCruz09  | 5  | 2  | 4  | 2  |

According to the result of HerboldMethod, Nam15 outperforms the other techniques by achieving the first rank, whereas CamargoCruz09 performs worst. However, our evaluation shows that the ranks of all approaches not only vary within the configurations, but all except Nam15, have inconclusive ranks across the configurations as well.

6 Discussion

In this study, we show that defect prediction approaches might exhibit different performance when evaluated under different contexts. By using a subset of the Juarez data used in the benchmarking study of Herbold et al. (2018), we observed
Fig. 6 Variation in ranks of techniques evaluated using CC configuration. Each sub-figure represents a window size from (1 to 18), x-axis shows split point in time (1 to 18), y-axis shows the ranks of technique from (1 to 5), and K represents window size. Techniques: A=Amasaki15, W=Watanabe08, C=CamargoCruz09, N=Nam15, M=Ma12.
Fig. 7 Variation in ranks of techniques evaluated using IC configuration. Each sub-figure represents a window size from (1 to 18), x-axis shows split point in time (1 to 18), y-axis shows the ranks of technique from (1 to 5), and K represents window size. Techniques: A=Amasaki15, W=Watanabe08, C=CamargoCruz09, N=Nam15, M=Ma12.
Fig. 8 Variation in ranks of techniques evaluated using CI configuration. Each sub-figure represents a window size from (1 to 18), x-axis shows split point in time (1 to 18), y-axis shows the ranks of technique from (1 to 5), and K represents window size. Techniques: A=Amasaki15, W=Watanabe08, C=CamargoCruz09, N=Nam15, M=Ma12.
Table 7 | Standard deviation in ranks of techniques calculated using Mean Rank Score of AUC, F-Score and MCC metrics

| Technique      | CC  | IC  | CI  | II  |
|----------------|-----|-----|-----|-----|
| Amasaki15      | 1.16| 1.20| 1.09| 1.18|
| Watanabe08     | 1.08| 0.99| 1.12| 1.00|
| CamargoCruz09  | 1.12| 1.22| 1.12| 1.23|
| Nam15          | 0.88| 0.81| 0.86| 0.77|
| Ma12           | 1.11| 1.13| 1.10| 1.15|

Fig. 9 | Variation in ranks of techniques evaluated using II configuration. x-axis shows the split point in time (1 to 18), y-axis shows the ranks of technique from (1 to 5). Remember window size(K) does not matter in II. Techniques: A=Amasaki15, W=Watanabe08, C=CamargoCruz09, N=Nam15, M=Ma12.

a disagreement in the ranks reported in Herbold’s original study and HERBOLD-METHOD. We also explain in this paper that cross-validation is not an appropriate way of training defect prediction models because cross-validation randomly splits the data irrespective of its time order. This type of evaluation might lead to the training of models on future data, which is in practice, not available for use. As a result, the performance estimates of these models may also be biased, and under realistic settings the model may perform better or worse than the estimates produced by making unrealistic assumptions.

Previous studies engaged in time-travel because of a cross-validation based evaluation, therefore to avoid it, we adopt a time-aware evaluation, and in Table 6 we report the new rankings of five techniques evaluated over four configurations. A comparison of our resulting ranks with the ranks reported by [Herbold et al., 2018] and the ranks obtained from HERBOLD-METHOD suggest that defect prediction models yield different conclusions when evaluated using time-aware evaluation on data from different time periods.

Although, our study is limited to the area of defect prediction, the time-aware methodology employed in this paper can be used to evaluate the conclusion stabil-
ity of other software analytic approaches, such as duplicate bug report prediction, effort estimation, and bad smell detection. To this end, our experimental results ascertain that our concern about over generalization of conclusions is legitimate. In our evaluation, which is based on the four time-aware configurations, the ranks of techniques vary by +1 or -1 within the configurations as well as across them. Only Nam15 achieved the same rank in all four configurations and in the HERBOLD-METHOD. The other techniques degrade by 2 or 3 ranks in certain configurations, which means that there is no agreement and thus high instability in the remaining four ranks. On a side note, these configurations allow for a systematic way of generating training and test data and also seem promising, as evaluations based on them exhibit diverse results which are realistically closer to the performance that a technique will yield in practice.

Lastly, it should be noted that the computational cost of training a large number of models corresponding to all configurations can be high, especially for models that employ sophisticated training techniques such as Neural Networks. Therefore, only some configurations or a few windows in each configuration can be used to achieve realistic performance estimates. Having said that, the choice of configuration entirely depends on the purpose of evaluation, as we explained in the methodology section. In either case, however, a time-stamped data set or version release dates are required to carry out the evaluation.

7 Threats to Validity

Construct Validity. We use the source code provided by [Herbold 2017a] for the evaluation. This poses a threat to the construct validity of our study but to counteract that, we also look into the original papers and make sure the implementations were correct.

External Validity. The external validity of the study is limited by the use of Jurezcko dataset. Our experiment relies on dates and timestamps which was not available in any of the publicly available datasets hence we relied on a single dataset for our study. The dataset has 20 metrics hence the results of our study might only hold for data having similar characteristics. In the future, we plan to evaluate more approaches using a variety of CPDP settings and data sets.

Internal Validity. The internal validity of the study suffers to a small extent due to reliance on the assumptions made in prior works. We have not tuned the hyperparameters of the decision tree but have instead relied on the evaluation settings similar to [Herbold et al. 2018]. An interesting future work is to examine the effect of tuning model parameters on the results.

8 Conclusion

Software engineering researchers often make claims about the generalization of the performance of their techniques outside the contexts of evaluation. In this paper we investigate whether conclusions in the area of defect prediction—the claims of the researchers—are stable throughout time.
We show lack of conclusion stability for multiple techniques when they are evaluated at different points in a project’s evolution. By following a time-aware methodology we found out that conclusions regarding ranking and performance of techniques replicated by Herbold et al. (2018) benchmarking study are not stable across different periods of time. With a standard deviation in F-Score of 0.1 or more, we find that context (the time) of evaluation can drastically change relative performance of defect prediction techniques evaluated, given the time frames, projects, and techniques we used for evaluation. Thus our results indicate huge variations in the performance of approaches depending on the time period of evaluation, as a consequence of which their rankings also changed.

This case study provides evidence that in the field of defect prediction the context of evaluation (in our case time) has a strong effect on the conclusions reached. Therefore it is imperative that empirical software engineering researchers do not over generalize their results but instead couch their claims of performance within the contexts of their evaluation—a field-wide faux pas that perhaps even this paper engages in.

References

Amasaki S, Kawata K, Yokogawa T (2015) Improving cross-project defect prediction methods with data simplification. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, pp 96–103, DOI 10.1109/SEAA.2015.25
Basili VR, Briand LC, Melo WL (1996) A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software engineering 22(10):751–761
Chidamber SR, Kemerer CF (1994) A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20(6):476–493
Cruz AE, Ochimizu K (2009) Towards logistic regression models for predicting fault-prone code across software projects. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement, pp 460–463, DOI 10.1109/ESEM.2009.5316002
D’Ambros M, Lanza M, Robbes R (2012) Evaluating defect prediction approaches: a benchmark and an extensive comparison. Empirical Software Engineering 17(4-5):531–577
Ekanayake J, Tappolet J, Gall HC, Bernstein A (2009) Tracking concept drift of software projects using defect prediction quality. In: 2009 6th IEEE International Working Conference on Mining Software Repositories, IEEE, pp 51–60
Ekanayake J, Tappolet J, Gall HC, Bernstein A (2012) Time variance and defect prediction in software projects. Springer, vol 17, pp 348–389
Fenton N, Neil M, Marsh W, Hearty P, Radlinski L, Krause P (2007) Project data incorporating qualitative factors for improved software defect prediction. In: Third International Workshop on Predictor Models in Software Engineering (PROMISE’07: ICSE Workshops 2007), pp 2–2, DOI 10.1109/PROMISE.2007.11
Fischer M, Pinzger M, Gall H (2003) Populating a release history database from version control and bug tracking systems. In: International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings., IEEE, pp 23–32
Hassan AE (2009) Predicting faults using the complexity of code changes. In: Proceedings of the 31st International Conference on Software Engineering, IEEE Computer Society, pp 78–88
Herbold S (2015) Crosspare: a tool for benchmarking cross-project defect predictions. In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), IEEE, pp 90–96
Herbold S (2017a) sherbold/replication-kit-tse-2017-benchmark: Release of the replication kit
Herbold S (2017b) A systematic mapping study on cross-project defect prediction. arXiv preprint arXiv:170506429
Herbold S, Trautsch A, Grabowski J (2018) A comparative study to benchmark cross-project defect prediction approaches. IEEE Transactions on Software Engineering 44(9):811–833, DOI 10.1109/TSE.2017.2724538
Hindle A, Onuekwo C (2019) Preventing duplicate bug reports by continuously querying bug reports. Empirical Software Engineering 24(2):902–936
Huang Q, Xia X, Lo D (2017) Supervised vs unsupervised models: A holistic look at effort-aware just-in-time defect prediction. In: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp 159–170
Jimenez M, Rwemalika R, Papadakis M, Sarro F, Le Traon Y, Harman M (2019) The importance of accounting for real-world labelling when predicting software vulnerabilities. In: Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)
Jureczko M, Madeyski L (2010) Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, ACM, New York, NY, USA, PROMISE ’10, pp 9:1–9:10, DOI 10.1145/1868328.1868342, URL http://doi.acm.org/10.1145/1868328.1868342
Kamei Y, Fukushima T, McIntosh S, Yamashita K, Uabayashi N, Hassan AE (2016) Studying just-in-time defect prediction using cross-project models. Empirical Software Engineering 21(5):2072–2106
Koru AG, Liu H (2005) An investigation of the effect of module size on defect prediction using static measures. SIGSOFT Softw Eng Notes 30(4):1–5, DOI 10.1145/1082983.1083172, URL http://doi.acm.org/10.1145/1082983.1083172
Krishna R, Menzies T (2018) Bellwethers: A baseline method for transfer learning. IEEE Transactions on Software Engineering
Lessmann S, Baesens B, Mues C, Pietsch S (2008) Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Transactions on Software Engineering 34(4):485–496
Ma Y, Luo G, Zeng X, Chen A (2012) Transfer learning for cross-company software defect prediction. Inf Softw Technol 54(3):248–256, DOI 10.1016/j.infsof.2011.09.007, URL http://dx.doi.org/10.1016/j.infsof.2011.09.007
Martin R (1994) Oo design quality metrics-an analysis of dependencies. In: Proc. Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA’94
Menzies T, Di Stefano JS (2004) How good is your blind spot sampling policy. In: Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings., pp 129–138, DOI 10.1109/HASE.2004.1281737
Menzies T, DiStefano J, Orrego A, Chapman R (2004) Assessing predictors of software defects. In: Proc. Workshop Predictive Software Models
Menzies T, Milton Z, Turhan B, Cukic B, Jiang Y, Bener A (2010) Defect prediction from static code features: current results, limitations, new approaches. Automated Software Engineering 17(4):375–407
Menzies T, Butcher A, Marcus A, Zimmermann T, Cok D (2011) Local vs. global models for effort estimation and defect prediction. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), IEEE, pp 343–351
Mockus A, Weiss DM (2000) Predicting risk of software changes. Bell Labs Technical Journal 5(2):169–180
Morasca S, Ruhe G (2000) A hybrid approach to analyze empirical software engineering data and its application to predict module fault-proneness in maintenance. Journal of Systems and Software 53(3):225–237
Nagappan N, Ball T (2005) Use of relative code churn measures to predict system defect density. In: Proceedings of the 27th international conference on Software engineering, ACM, pp 284–292
Nam J, Kim S (2015) Clami: Defect prediction on unlabeled datasets (t). In: Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE Computer Society, Washington, DC, USA, ASE ’15, pp 452–463, DOI 10.1109/ASE.2015.56, URL https://doi.org/10.1109/ASE.2015.56
Nam J, Pan SJ, Kim S (2013) Transfer defect learning. In: 2013 35th International Conference on Software Engineering (ICSE), IEEE, pp 382–391
Peters F, Menzies T, Marcus A (2013) Better cross company defect prediction. In: Proceedings of the 10th Working Conference on Mining Software Repositories, IEEE Press, pp 409–418
Rahman F, Devanbu P (2013) How, and why, process metrics are better. In: 2013 35th International Conference on Software Engineering (ICSE), IEEE, pp 432–441
Rakha MS, Bezemer C, Hassan AE (2018) Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. IEEE Transactions on Software Engineering 44(12):1245–1268, DOI 10.1109/TSE.2017.2755905
Romano J, Kromrey JD, Coraggio J, Skowronek J, Devine L (2006) Exploring methods for evaluating group differences on the nsse and other surveys: Are the t-test and cohensd indices the most appropriate choices. In: annual meeting of the Southern Association for Institutional Research, Citeseer, pp 1–51
´Sliwerski J, Zimmermann T, Zeller A (2005a) When do changes induce fixes? In: ACM sigsoft software engineering notes, ACM, vol 30, pp 1–5
´Sliwerski J, Zimmermann T, Zeller A (2005b) When do changes induce fixes? In: ACM sigsoft software engineering notes, ACM, vol 30, pp 1–5
Tan M, Tan L, Dara S, Mayeux C (2015) Online defect prediction for imbalanced data. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, IEEE, vol 2, pp 99–108
Tang MH, Kao MH, Chen MH (1999) An empirical study on object-oriented metrics. In: Proceedings sixth international software metrics symposium (Cat. No. PR00403), IEEE, pp 242–249
Tantithamthavorn C, McIntosh S, Hassan AE, Ihara A, Matsumoto K (2015) The impact of mislabelling on the performance and interpretation of defect prediction models. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, IEEE, vol 1, pp 812–823
Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2017) An empirical comparison of model validation techniques for defect prediction models. IEEE Transactions on Software Engineering 43(1):1–18
Tantithamthavorn C, McIntosh S, Hassan AE, Matsumoto K (2018) The impact of automated parameter optimization on defect prediction models. IEEE Transactions on Software Engineering
Turhan B (2012) On the dataset shift problem in software engineering prediction models. Empirical Software Engineering 17(1-2):62–74
Turhan B, Menzies T, Bener AB, Di Stefano J (2009) On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering 14(5):540–578
Watanabe S, Kaiya H, Kaijiri K (2008) Adapting a fault prediction model to allow inter languagereuse. In: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, ACM, New York, NY, USA, PROMISE ’08, pp 19–24, DOI 10.1145/1370788.1370794, URL http://doi.acm.org/10.1145/1370788.1370794
Witten IH, Frank E, Hall MA, Pal CJ (2016) Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann
Yang X, Lo D, Xia X, Zhang Y, Sun J (2015) Deep learning for just-in-time defect prediction. In: 2015 IEEE International Conference on Software Quality, Reliability and Security, IEEE, pp 17–26
Yang Y, Zhou Y, Liu J, Zhao Y, Lu H, Xu L, Xu B, Leung H (2016) Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM, pp 157–168
Zhang F, Mockus A, Keivanloo I, Zou Y (2014) Towards building a universal defect prediction model. In: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, pp 182–191
Zimmermann T, Nagappan N (2007) Predicting subsystem failures using dependency graph complexities. In: The 18th IEEE International Symposium on Software Reliability (ISSRE’07), IEEE, pp 227–236
Zimmermann T, Premraj R, Zeller A (2007) Predicting defects for eclipse. In: Third International Workshop on Predictor Models in Software Engineering (PROMISE’07: ICSE Workshops 2007), IEEE, pp 9–9
Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ACM, pp 91–100