Local versus Global Models for Effort-aware Defect Prediction

Mariam El Mezouar
School of Computing
Queen's University
Kingston, Ontario, Canada
mariam@cs.queensu.ca

Feng Zhang
School of Computing
Queen's University
Kingston, Ontario, Canada
feng@cs.queensu.ca

Ying Zou
Department of Electrical and Computer Engineering
Queen's University
Kingston, Ontario, Canada
ying.zou@queensu.ca

ABSTRACT

Software entities (e.g., files or classes) do not have the same density of defects and therefore do not require the same amount of effort for inspection. With limited resources, it is critical to reveal as many defects as possible. To satisfy such need, effort-aware defect prediction models have been proposed. However, the performance of prediction models is commonly affected by a large amount of possible variability in the training data. Prior studies have inspected whether using a subset of the original training data (i.e., local models) could improve the performance of prediction models in the context of defect prediction and effort estimation in comparison with global models (i.e., trained on the whole dataset). However, no consensus has been reached and the comparison has not been performed in the context of effort-aware defect prediction.

In this study, we compare local and global effort-aware defect prediction models using 15 projects from the widely used AEEEM and PROMISE datasets. We observe that although there is at least one local model that can outperform the global model, there always exists another local model that performs very poorly in all the projects. We further find that the poor performing local model is built on the subset of the training set with a low ratio of defective entities. By excluding such subset of the training set and building a local effort-aware model with the remaining training set, the local model usually underperforms the global model in 11 out of the 15 studied projects. A close inspection on the failure of local effort-aware models reveals that the major challenge comes from defective entities with small size (i.e., few lines of code), as such entities tend to be correctly predicted by the global model but missed by the local model. Further work should pay special attention to the small but defective entities.

1. INTRODUCTION

Defects in a software system increase the maintenance cost. It is estimated that fixing defects consumes between 50-80% of the software development resources of companies [11]. It is vital to the success of a software organization to maximize the utilization of the limited resources for developing and testing. To this end, defect prediction models have been extensively studied [19, 5, 29, 44].

Due to the limited resources for developing and testing software, it may be impractical to inspect all files that are predicted as defective [38, 1, 2, 3]. Effort-aware[1] defect prediction models are therefore proposed [16, 26, 37], aiming to reduce the candidate set of files to inspect. Effort-aware defect prediction models rank entities based on the predicted defect density (i.e., the ratio of the number of predicted defects in a file to the number of lines of code in the file) so that developers could select the entities with the highest defect density for further inspection (e.g., code reviews). For example, given two classes *MainPreferencePage* (71 lines of Java code) and *ProductExportOperation* (613 lines of Java code) from the Eclipse PDE project, the number of predicted defects in these two classes are three and six, respectively; a traditional defect prediction model would suggest to give a higher priority to the class *ProductExportOperation* as it has more predicted defects, but an effort-aware model would rank the class *MainPreferencePage* higher because its defect density is greater (i.e., 0.04 versus 0.01). Indeed, it is more efficient to review 71 lines of code and detect three defects than to review 671 lines of code to locate six defects.

Software engineering data (e.g., source code descriptive metrics) is clearly crucial to train high quality defect prediction models. Besides studies investigating how to remove noise from the training data [20, 21], researchers have also looked into using local regions of the training data that have similar properties to build models (i.e., local models) [7, 27]. While prior studies have found that local models could result in a better fit to the underlying data [7], they also find that local models show only small improvements over global models, with respect to prediction errors. In this study, we complement this line of work by further investigating local regions in the training data in terms of their predictive performance and characteristics, in the context of effort-aware defect prediction. To the best of our knowledge, no prior study has looked into the suitability of local models for the task of effort-aware defect prediction modelling.

We conduct an empirical study using five projects from AEEEM dataset [12] and ten projects from PROMISE dataset [15]. To obtain the training for building local models, we apply the off-the-shelf clustering method (i.e., partitioning around medoids [34]). Local prediction models are built within each cluster. We investigate the variance in the performance of the local models and compare local models with global models. We then study the characteristics of their associated clusters. Accordingly, we investigate the following two research questions:

(RQ1) How do local models compare to global models, with

[1]Same as the work by Mende and Koshke [26], this study uses lines of code as a proxy of the effort for inspecting a particular file.

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Table 1: Description of the 15 subject projects.

Project name	Project description
Ant (ant.apache.org)	Ant is a known Java-based, shell independent build tool.
Camel (camel.apache.org)	Apache Camel is a powerful open source integration framework.
Ivy (ant.apache.org/ivy)	Ivy is a dependency manager focusing on flexibility and simplicity.
Jedit (www.jedit.org)	jEdit is a mature programmer's text editor.
Log4j (logging.apache.org/log4j)	Log4j is a well known logging framework.
POI (poi.apache.org)	The POI project consists of APIs for manipulating various file formats based upon Microsoft's OLE 2 Compound Document format, and Office OpenXML format.
Tomcat (tomcat.apache.org)	Apache Tomcat is an open source software implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies.
Xalan (xml.apache.org/xalan-j)	Xalan is an XSLT processor for transforming XML documents into HTML, text, or other XML document types.
Xerces (xerces.apache.org/xerces-j)	Xerces is a Parser that supports the XML 1.0 recommendation and contains advanced parser functionality.
Eclipse JDT Core (www.eclipse.org/jdt/core)	Eclipse JDT Core provides the tool plug-ins that implement a Java IDE supporting the development of any Java application, including Eclipse plug-ins.
Eclipse PDE UI (www.eclipse.org/pde/pde-ui)	Eclipse PDE UI provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments, features, update sites and RCP products.
Equinox Framework (www.eclipse.org/equinox)	Equinox is an implementation of the OSGi core framework specification, a set of bundles that implement various optional OSGi services and other infrastructure for running OSGi-based systems.
Mylyn (www.eclipse.org/mylyn)	Mylyn is a Task-Focused Interface for Eclipse that reduces information overload and makes multi-tasking easy.
Apache Lucene (lucene.apache.org)	Lucene provides Java-based indexing and search technology, as well as spellchecking, hit highlighting and advanced analysis/tokenization capabilities - Present in both AEEEM and Promise datasets.

respect to prediction performance?

In this question, we explore the performance of local defect prediction models (i.e., built upon a cluster of files) and global defect prediction models (i.e., built upon the original dataset of the entire project). We observe that there always exists one local defect prediction model that yields poor performance. Moreover, we find that such cluster usually has the characteristics that they have a lower ratio of defective files, a smaller size of files, and lower code complexity.

(RQ2) Can we improve the performance of effort-aware defect prediction by training the model using only the high performing cluster?

In most of the studied projects (i.e., 11 out of 15), our results show a superiority of the global effort-aware models trained on the whole training data; thus showing that larger and more diversified training samples are better suited to train effort-aware defect prediction models. We further investigate the reasons behind the superiority of the global model. We find that the local model marks many of the small but defective files as clean. Such misclassification significantly downgrades the performance of local effort-aware defect prediction models, as the small defective files require few effort for exposing defects.

As a summary, the major contribution of this work is three-fold:

- Conduct comparisons on local and global models in the context of effort-aware defect prediction using different clustering algorithms than prior studies, with further understanding on the characteristics of the local clusters. Although some local defect prediction models can outperform the global defect prediction models, there always exists at least one local defect prediction model that fails to compete with the global defect prediction model.
- Explore the possibility of building effort-aware defect prediction models using the only the high performing cluster. We find that the global model is generally superior.
- Investigate the reasons behind the superiority of the global model. Our investigation reveal that local models generally misclassify the small defective files, while such files are usually classified correctly by the global model trained on a more varied dataset in terms of metrics' values.

Paper organization. We present the overview of our case study design in Section 2. The results are described in Sections 3. Section 4 presents the threats to validity. Finally, the related work and conclusions are presented in Sections 5 and 6, respectively.

2. CASE STUDY DESIGN

This section presents the design of our case study. First, we introduce our subject projects and data pre-processing methods. Then, we describe the performance measures and the evaluation approach of the defect prediction models.

2.1 Data Collection

In this study, we use 15 projects (see Table 1) from two publicly available datasets that have been widely used in defect prediction studies [8][33][23]. We use ten projects of the PROMISE dataset [15] and all the five projects of the AEEEM dataset [12]. The programming language of all projects is Java. We show in Table 2 the descriptive statistics of the projects under study.

2.2 Software Metrics

To build local models trained on subsets of the data with similar properties, we measure the similarity among software entities and group the similar entities into the same cluster. We use software metrics to measure the similarity between software entities. The AEEEM dataset has both process and product metrics; and the PROMISE dataset has only product metrics.

Product metrics characterize various aspects of a software product. Some of the best-known product metrics for object oriented projects are Chidamber & Kemerer object-oriented metrics [6] that consist of six metrics. Table 3 shows the product metrics that are provided in both datasets and used in this work.

To better understand the various aspects of a software product that are measured using the product metrics, we further classify the product metrics into five categories (i.e., complexity, coupling, cohesion, abstraction, and encapsulation) based on the work by Zhang *et al.* [42]. We refer the interested reader to the work by D'Ambros *et al.* [12] and Jureczko *et al.* [15] for more detailed

Table 2: Descriptive statistics of the 15 subject projects

Dataset	Subject project	Release	# Instances	# Defects
PROMISE	Ant	1.7	745	338
	Camel	1.6	965	500
	Ivy	1.4	241	18
	Jedit	4.0	306	226
	Log4j	1.0	135	61
	Lucene	2.2	247	414
	Poi	3.0	442	500a
	Tomcat	6.0	858	114
	Xalan	2.3	885	625
	Xerces	1.3	453	193
AEEEM	Eclipse JDT Core	3.4	997	463
	Eclipse PDE UI	3.4.1	1,592	401
	Equinox framework	3.4	439	279a
	Mylyn	3.1	2,196	677
	Apache Lucene	2.4.0	691	103

information about the product metrics included in the PROMISE and AEEEM datasets.

Process metrics are collected over a long period of time to capture how developers make modifications to the source code. The modifications are usually performed to fix defects or to implement new features. Source code and logs stored in a revision control system are the main sources to extract such metrics. Process metrics have been widely applied to build defect prediction models [12]. Table 4 shows the process metrics that are included in the AEEEM dataset. More details of these metrics are described in the paper by D'Ambros *et al.* [12].

In the presence of multicollinearity, the estimate of the impact of one metric on the dependent variable tends to be less precise, thus weakening the prediction model. To mitigate the multicollinearity, we conduct a correlation analysis using Spearman's rank coefficient. We opt for Spearman's rank correlation test over other non-rank correlation tests (e.g., Pearson) because rank correlation is more robust when dealing with data that is not normally distributed [41]. Among each group of highly correlated metrics (Spearman's $\rho >$ 0.7 [41]), we select only one metric from the group to include in the model.

Upon the extraction of the descriptive statistics for the metrics namely the mean, standard deviation, median, min, max, and skewness, we notice a relatively high amount of skewness. To mitigate the skewness effects during modelling, we apply the log transformation *log(x+1)* to each metric [30].

2.3 Clustering for Building Local Models

Prior to build local models, we apply a clustering algorithm on the dataset to obtain the subset of the training data. We use the k-medoids algorithm, a more robust version of K-means [34]. K-means clustering iteratively finds the k centroids and assigns every object to the nearest centroid, more specifically the coordinate of each centroid is the mean of the coordinates of the objects in the cluster. Unfortunately, K-means clustering is known to be sensitive to the outliers. For this reason, K-medoids clustering considers representative objects, called medoids, instead of centroids. K-medoids clustering is less sensitive to outliers in comparison with the K-means clustering because K-medoids clustering is based on the most centrally located object in a cluster. Among many algorithms for K-medoids clustering, partitioning around medoids (PAM) proposed by Kaufman and Rousseeuw [18] is known to be the most powerful. We use an implementation of the k-medoids algorithm available in R under the package $cluster$. The library function is called pam and takes the dataset and the number of clusters to be generated as its arguments.

Table 3: List of product metrics that are selected for the projects of each dataset

Dataset	Category	Product Metric
PROMISE	Complexity	Lines of Code (LOC)
		Weighted Methods per Class (WMC)
		Number of Public Methods (NPM)
		Average Method Complexity (AMC)
		Max McCabe's Cyclomatic Complexity (Max_cc)
		Avg McCabe's Cyclomatic Complexity (Avg_cc)
		Measure of Aggregation (MOA)
	Coupling	Coupling between object classes (CBO)
		Response of a Class (RFC)
		Afferent Couplings (CA)
		Efferent Couplings (CE)
		Inheritance Coupling (IC)
		Coupling Between Methods (CBM)
	Cohesion	Lack of cohesion in methods (LCOM)
		Lack of cohesion in methods (LCOM3)
		Cohesion Among Methods of Class (CAM)
	Abstraction	Depth of Inheritance Tree (DIT)
		Number Of Children (NOC)
		Measure of Functional Abstraction (MFA)
	Encapsulation	Data Access Metric (DAM)
AEEEM	Complexity	numberOfAttributes
		numberOfAttributesInherited
		numberOfLinesOfCode)
		numberOfMethods)
		numberOfMethodsInherited)
		numberOfPrivateAttributes)
		numberOfPrivateMethods)
		numberOfPublicAttributes)
		numberOfPublicMethods)
		Weighted Methods per Class (WMC)
	Coupling	Coupling between Object Classes (CBO)
		Number of Input Data (fanIn)
		Number of Input Data (fanOut)
		Response of a Class (RFC)
	Cohesion	Lack of cohesion in methods(LCOM)
	Abstraction	Depth of InheritanceTree (DIT)

2.4 Performance Measures of Prediction Models

Defect prediction. Defect prediction models are used to predict the number of defects in software entities, based on their descriptive metrics. To evaluate the defect prediction performance, we use the prediction error $abs(Y_{actual} - Y_{predicted})$, similarly to Bettenburg *et al.* [7]. This evaluation measure informs us how far off the predicted value is from the actual value. The closer the prediction error is close to 0, the better the performance of the defect performance model is.

Effort-aware defect prediction. We use effort-aware defect prediction to predict the defect density in software entities. To evaluate the performance of the effort-aware defect prediction, we use a measure called P_{opt}, where *opt* stands for optimal, similarly to Mende *et al.* [26]. We compute the difference in terms of area (Δ_{opt}) between the two lines representing the LOC-based cumulative lift charts of the optimal and prediction models, as illustrated in Figure 1.

The values on the x-axis represent the percentages of lines of code; while the y-axis shows the percentages of defects contained in the corresponding number of lines of code. The line representing the actual model shows the cumulative values of the LOC of files sorted by their decreasing *actual* fault density ($\frac{ActualDefects}{LOC}$). The line representing the predicted model reflects the cumulative values of the LOC of files sorted by their decreasing *predicted* fault density ($\frac{PredictedDefects}{LOC}$).

Table 4: List of process metrics that are selected for the projects of the AEEEM dataset

Dataset	Category	Process Metric
PROMISE		None
AEEEM	Previous defects	numberOfBugsFoundUntil numberOfNonTrivialBugsFoundUntil numberOfMajorBugsFoundUntil numberOfCriticalBugsFoundUntil numberOfHighPriorityBugsFoundUntil
	Change metrics	numberOfVersionsUntil numberOfFixesUntil numberOfRefactoringsUntil numberOfAuthorsUntil linesAddedUntil maxLinesAddedUntil avgLinesAddedUntil linesRemovedUntil maxLinesRemovedUntil avgLinesRemovedUntil codeChurnUntil maxCodeChurnUntil avgCodeChurnUntil ageWithRespectTo weightedAgeWithRespectTo
	Entropy	CvsEntropy CvsWEntropy CvsLinEntropy CvsLogEntropy CvsExpEntro

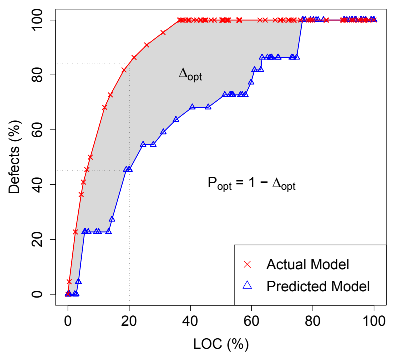


Figure 1: An illustrating example of LOC-based cumulative lift chart

We normalize all values to the range between 0% and 100%. The chart indicates the percentage of lines of code to be reviewed to cover a certain amount of defects. For instance, reviewing 20% of the lines of code allows to cover 45% of the defects in the predicted model, and over 80% of defects in the actual model, as shown by the dashed lines in Figure 1. Larger P_{opt} value (closer to 1) indicates a smaller difference between the actual and predicted models (as shown in Equation 1). A smaller area between the two lines (smaller P_{opt}) in the chart indicates that the predicted model is closer to the actual model.

$$P_{opt} = 1 - \Delta_{opt} \tag{1}$$

2.5 Model Evaluation

To account for random observation bias, we do the following for every experiment. First, the cross-validation process is repeated 10 times (i.e., the folds). In each cross-validation, the 10 sub-samples used exactly once as the testing data. The 10 results from the folds are averaged to produce a single prediction performance of the model. We repeat each experiment 10 times (10 cross validation repeated 10 times) to get the most possible stable results and we calculate the average of the results for a single estimation of the performance. Overall, we perform a 100 times cross-validation. The evaluation approach is illustrated in Figure 2.

3. RESULTS

In this section, we present the results of our approach with respect to our two research questions.

RQ1. How do local models compare to global models, with respect to prediction performance?

Motivation. A study by Menzies *et al.* [27] shows that there lies some benefit in partitioning software engineering data into smaller clusters with similar characteristics. The study [27] showed that local models (trained on clusters of data) lead to better fits compared to global models (trained on the entire data). Bettenburg *et al.* [7] extended the work of Menzies *et al.* [27] and found that local models lower the prediction error of prediction models. We use a similar approach as Bettenburg *et al.* [7] to build local prediction models. However, we use different clustering algorithms to partition the data (i.e., PAM and spectral clustering). Besides, Bettenburg *et al.* [7] only use two projects from the PROMISE repository (Xalan 2.6 and Lucene 2.4), we include in our study 10 projects from the PROMISE dataset and 5 projects from the AEEEM dataset. We also attempt to gain more understanding about the local models, by assessing their individual defect prediction performances and their characteristics. This study focuses on effort-aware prediction; however, for this first RQ, we examine the quality of local models in terms of the traditional defect prediction to gain general insights.

Approach. To answer this research question, we apply the following approach. First, we build a global defect prediction model trained on the full dataset to predict the number of defects. To predict the number of defects in each file, we build a linear regression model, a commonly applied approach to model the relationship between a dependent variable (i.e., number of defects) and a set of independent variables (i.e., software metrics) [43][12][7]. We assess the performance of the global model using the *prediction error* measure (see Section 2.4)

Second, we cluster the data and build local defect prediction models based on each cluster. The local defect prediction models are built and evaluated using the same approach as the global defect prediction model. Lastly, we compare the performance of the global defect prediction model and local defect prediction models associated with each project. The lower the *prediction error* of the model, the more accurate it is.

Clustering the data is a fundamental steps in building the local defect prediction models, because the clustering possibly impacts the quality of the derived local models. Since the clustering method we use has only one parameter (i.e., the number of clusters K), we perform a sensitivity analysis by varying the number of clusters (i.e., K between 2 to 25) to assess the stability of our findings.

We test the following hypothesis:
H_{01}: *the global and the local defect prediction models associated to a project have equal performances.*

To test this hypothesis, we apply the Kruskal-Wallis H test [24] to assess whether our hypothesis holds across the 15 projects. The Kruskal-Wallis H test is a non-parametric statistical method to assess whether two or more distributions have statistically significant differences. The advantage of using non-parametric statistical methods is that they make no assumptions about the distribution of the data. Also, since we carry multiple comparisons, we correct the

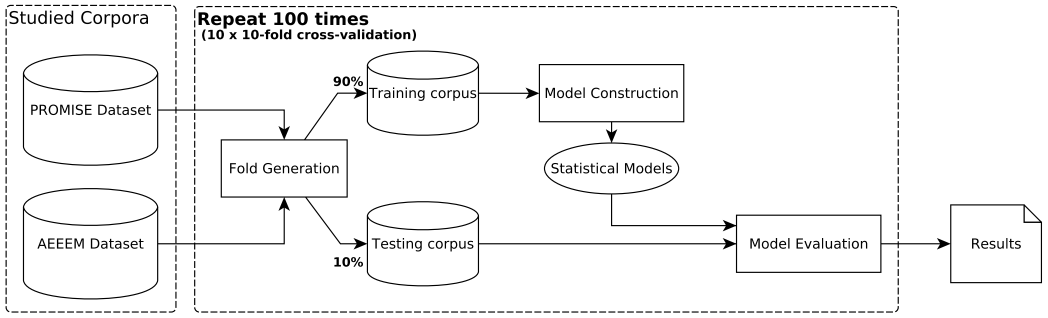


Figure 2: An overview of our evaluation approach

p-value using the Holm-Bonferroni method [14].

To identify the main characteristics that differentiate the clusters on which the local models are trained, we collect the values of each metric pertaining to all the files in a cluster. For each cluster in every project, we obtain a distribution of each metric. A metric distribution in a given cluster contains the metric values of the files in that cluster. We perform a Kruskal-Wallis H test between the distributions of each metric in a project to determine whether the distributions are significantly different and therefore determine whether the metric is significant in separating the different clusters. In addition to assessing the significance of the difference between distributions, we also measure the "size" of the difference using Cliff's delta [36]. Cliff's delta is non-parametric, thus it does not make assumptions about the distribution of the data. It is reported to be more robust than Cohen's d [36]. Cliff's delta reflects the degree of overlap between different distributions. It ranges from -1 (if all values in the first distribution are larger than the second) to +1 (all values in the first distribution are smaller than the second). When the two distributions are equal, it is equal to 0 [9]. The results of Cliff's delta can be described with Cohen's d standards (small, medium, and large). A *medium* effect is described by [10] as a difference noticeable by a careful observer; while a *large* effect is more visible than a medium one. We characterize each cluster by providing representative values of its significant metrics.

Findings. Similarly to the prior studies comparing local and global models in defect prediction [7][26], we also observe the potential benefit of local models. In fact, at least one local model in each project outperforms or has comparable performance to the global model, in terms of the prediction error. This finding confirms the possible benefit of training defect prediction models that are tailored to regions of the data with similar properties. We show in Figure 3 the results of the experiment for the measure *prediction error* for all the projects. We only show the results of K=2 clustering to simplify the presentation of the results. For each project, we show the performance of the global model and of the two local models associated with K=2 clustering. The order of the two local models is randomly assigned In the case of K=2 clustering, one of the local models outperforms the global model in terms of prediction error in 11 out of 15 projects, and has comparable performance to the global model in the remaining 4 projects.

However, there always exist a local model that significantly underperforms the global defect prediction model and the other local defect prediction models, except for POI and Xerces. The performance of different local defect prediction models is not equal. This finding suggests that the local models are not always able to compete with the global model. This observation is consistent across the 15 projects. Our findings lead us to reject the hypothesis

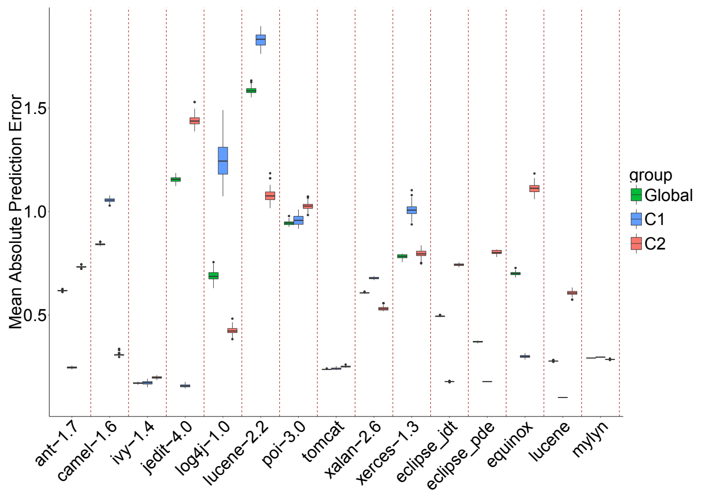


Figure 3: Prediction error of global models and local models of clusters (K = 2)

H_{01} and conclude that local models do not have equal predictive performances. We further explore the ratio of defective files in each cluster and find that local models that experience poor performance (i.e., higher prediction error) are built on the clusters of files that have a much lower ratio of defective files. Table 5 shows a comparison of the ratios of defective files in each cluster of the 15 projects. The lower ratio of defective files in the clusters whose local models perform poorly could possibly be one reason behind their failure.

Moreover, we notice that the drawn conclusions are similar for all Ks (i.e., 2 to 25) that we tested in our sensitivity analysis. As a matter of fact, there is always *at least* one local defect prediction model that performs very poorly compared to the rest.

In the remaining of the paper, we refer to the clusters whose models outperform or have comparable performance to the global models as high performing clusters. The clusters whose models underperform the global models are called low performing clusters.

We find that the low performing clusters in every project from both datasets are composed of smaller and less complex files. In the AEEEM dataset that also contains process metrics, files in the low performing clusters have on average less past defects, fewer authors, less revisions and less churn. To illustrate the characteristics of the low performing clusters, we show in Table 6 the average and standard deviation values for each significant metric across each dataset for K=2 clustering. We also show the corresponding values of the high performing cluster. The selected metrics shown

Table 5: The ratio of defective files in the entire set of files, and in each cluster of files

Dataset	Subject project	Overall	Low performing cluster	High performing cluster
PROMISE	Ant v1.7	22.29%	5.75%	38.16%
	Camel v1.6	19.48%	15.38%	25.45%
	Ivy v1.4	6.63%	2.27%	11.93%
	Jedit v4.0	24.51%	12.84%	53.41%
	Log4j v1.2	25.37%	5.00%	33.68%
	Lucene v2.4	58.30%	52.53%	81.63%
	POI v3.0	63.57%	32.45%	79.73%
	Tomcat v6	8.97%	2.50%	17.20%
	Xalan v2.6	46.44%	20.39%	60.07%
	Xerces v1.3	15.23%	4.02%	24.02%
AEEEM	Eclipse JDT v2.4	22.71%	8.16%	26.88%
	Eclipse PDE	13.96%	6.07%	20.44%
	Equinox	39.81%	13.49%	56.57%
	Lucene v2.6	9.26%	4.31%	12.16%
	Mylyn v1.3	13.16%	12.79%	14.54%

in Table 6 are those that experience significant distributions across all projects. In the AEEEM dataset, 23 metrics significantly distinguish the low and high performing clusters. In the PROMISE dataset, 15 metrics significantly distinguish the clusters. We analyze the significant metrics in both datasets to describe the files in the low and high performing clusters.

Overall, we conclude that the files associated with the low performing clusters are simpler in terms of code complexity, compared to the files in the high performing cluster. They also have a less complex history in terms of past defects, authors, and churn.

The low performing clusters when clustering with higher K's are all roughly part of the lowest performing cluster when using K=2. We investigate the impact of using different number of clusters when building local defect prediction models. We use Principal Component Analysis (PCA) to visualize the clusters generated when we use different K's to cluster the data. PCA is a widely used method to explore high-dimensional data [35], using a 2-D scatterplot with two principal components. The PCA plot shown in Figure 4 shows a scatterplot with axes corresponding to the two first principal components that show the largest variance.

We use the Ant project as an illustrating example. The sub-figures show the results of values of K from 2 to 5 for the Ant Project. The visualization of the PCA plot becomes harder to decipher with K values higher than 5. In Figures 4-a, b, c and d, the low performing cluster is indicated with red triangles. As shown in Figure 4, the low performing clusters in K=3, 4 and 5 are all roughly part of the lowest performing cluster in K=2. Clustering with a small K produces larger clusters allowing us to identify as many data points as possible that have lower predictive performance. Since the low performing model in K=2 clustering performs significantly less than the global and the other local models, we focus on clustering the data into 2 clusters in the remaining of the paper.

At least one local model in each project has a very low predictive performance in terms of prediction error. Moreover, the clusters associated with the low performing local models have very low ratio of defective files compared to the global model, thus indicating a possible reason behind the failure of these local models.

RQ2. Can we improve the performance of effort-aware defect prediction by training the model using only the high performing cluster?

Motivation. We investigate whether we can improve the prediction performance of effort-aware defect prediction models by training a local model instead of a global model. Given that not all clusters in a dataset have equal predictive performances as shown in RQ1, we explore in this research question the impact of training the effort-aware model using only the high performing cluster. We investigate whether the local effort-aware defect prediction model trained on the high performing cluster can achieve better performance than the global effort-aware defect prediction model.

Table 6: Average of the statistics values of the significant metrics across each dataset. The selected metrics for each dataset are significant in all projects of that dataset.

Dataset	Metric name	Low performing cluster	High performing cluster
		Avg. ± Std.	Avg. ± Std.
AEEEM	numBugsFoundU	1.75 ± 1.97	10.53 ± 15.30
	numNTBugsFoundU	1.41 ± 1.68	9.09 ± 12.96
	numOfVersionsUntil	8.02 ± 5.94	36.57 ± 37.75
	numOfAuthorsUntil	2.83 ± 1.45	5.50 ± 2.21
	linesAddedUntil	53.52 ± 52.50	1003.81 ± 2143.65
	maxLinesAddedUntil	20.54 ± 21.71	206.40 ± 296.21
	linesRemovedUntil	44.73 ± 55.88	787.30 ± 1816.43
	maxLinesRemovedUntil	19.22 ± 26.56	184.20 ± 294.52
	avgLinesRemovedUntil	3.63 ± 3.91	13.95 ± 16.11
	CvsEntropy	3.96 ± 2.64	11.59 ± 7.32
	CvsWEntropy	0.01 ± 0.01	0.10 ± 0.20
	CvsLinEntropy	0.02 ± 0.03	0.07 ± 0.05
	CvsLogEntropy	0.72 ± 1.30	2.11 ± 1.68
	CvsExpEntropy	0.04 ± 0.05	0.12 ± 0.09
	cbo	5.03 ± 4.81	15.78 ± 20.14
	fanOut	2.99 ± 3.25	9.40 ± 8.25
	lcom	25.39 ± 69.61	328.00 ± 1798.03
	numberOfLinesOfCode	71.02 ± 136.75	228.13 ± 368.73
	numberOfMethods	5.66 ± 4.86	15.15 ± 17.51
	numberOfPublicMethods	3.74 ± 3.47	9.23 ± 13.53
	rfc	20.68 ± 26.73	87.58 ± 129.27
	wmc	16.00 ± 27.49	57.37 ± 89.77
PROMISE	wmc	7.81 ± 6.44	18.42 ± 20.11
	dit	2.02 ± 1.31	2.16 ± 1.35
	noc	0.35 ± 1.54	0.85 ± 3.49
	cbo	8.03 ± 12.52	16.15 ± 21.39
	rfc	21.27 ± 17.63	50.22 ± 43.96
	lcom	68.96 ± 293.00	263.53 ± 977.51
	ca	4.39 ± 12.13	8.70 ± 19.82
	ce	2.65 ± 2.59	7.97 ± 7.40
	npm	6.27 ± 5.99	12.91 ± 14.04
	lcom3	1.09 ± 0.70	0.93 ± 0.44
	loc	191.61 ± 231.70	543.66 ± 886.53
	dam	0.53 ± 0.46	0.73 ± 0.37
	moa	0.64 ± 1.25	1.36 ± 1.89
	mfa	0.40 ± 0.40	0.36 ± 0.37
	cam	0.54 ± 0.22	0.35 ± 0.15

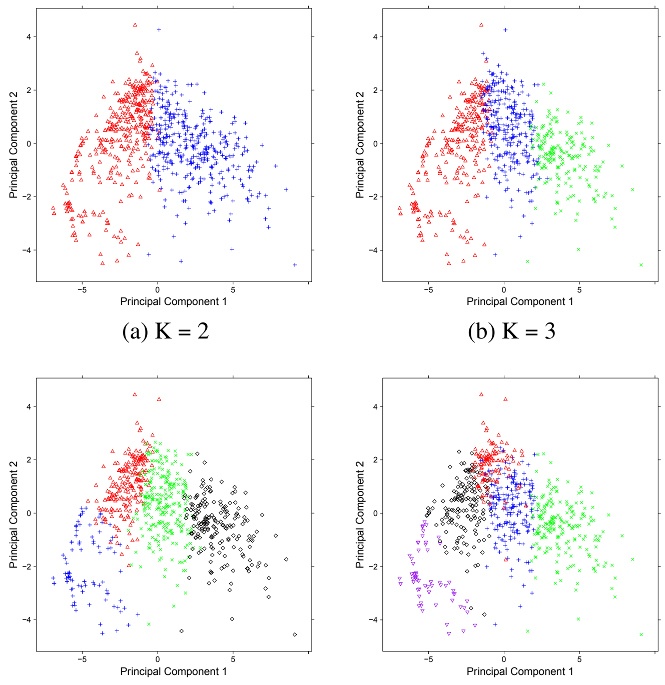
Approach. Similarly to RQ1, we build defect prediction models to predict the number of defects in each file. We then compute the predicted defect density of files and build the LOC-based cumulative lift charts of the actual model and the predicted model. Finally, we compute the area between the actual and predicted model to evaluate the performance of the effort-aware defect prediction models.

To build an effort-aware defect prediction model, we apply the same approach as Mende *et al.* [26]. This model uses the number of lines of code, as a measure of effort. The defect density of a file is defined as:

$$DefectDensity(x) = \frac{numberOfDefects(x)}{LOC(x)} \tag{2}$$

where $numberOfDefects(x)$ is the number of defects in a file x and $LOC(x)$ is the number of lines of code of the file. To predict the number of defects in each file, we build linear regression models, similarly to RQ1.

We aim to test the following hypothesis:



(a) K = 2 (b) K = 3

(d) K = 4 (e) K = 5

Figure 4: Visualization of the Clusters using the two first components of Principal Component Analysis

H_{02}: *Building the effort-aware defect prediction model using the high performing cluster improves the P_{opt} of the model.*

To assess the statistical significance of using the high performing cluster versus the entire dataset to build the prediction model, we perform the Kruskal-Wallis H test [24]. If the distributions are statistically different (p-value < 0.05), we conclude that prediction models built on the high performing clusters perform significantly better or worse than the original global model. To quantify the impact of the observed improvement, we calculate the effect size using Cliff's delta [36].

Findings. The effort-aware defect prediction model trained on the whole training set (i.e., global model) is superior in 11 out of the 15 studied projects. We report in Figure 5 the results for all the projects. Each subgroup in the figure corresponds to a project for which we show P_{opt} values of the global and high performing local models. The box plots shows a superiority of the global model built on the whole training set for ≈73% of the projects. We confirm the superiority of the global effort-aware defect prediction model using the Kruskal-Wallis H test [24]. We also report in Table 7 the difference in the performance between the global and local effort-aware defect prediction models. In the three projects where the local model outperforms the global model, we observe that all three projects (i.e., Ivy, Tomcat, and Lucene 2.6) have less than 10% of defective files. We can also observe that the low performing cluster for these three projects has less than 5% of defective files (see Table 5). In one project (i.e., Camel), the performance of the two models is close to each other.

The global model is better at predicting the bugginess of the files that are smaller in size, because it is trained on a more diversified training set in terms of metrics' values. The local model, on the other hand, misclassifies many of the smaller defective files as clean, which affects the performance of the effort-aware defect prediction performance. We carry an in-depth analysis of the cases (i.e., files) that cause the global model to have a higher performance, we find that the difference is caused by the local model's inability to recognize that some of the smaller files might be defective. The local model is trained with a subset of the data containing mostly long and more complex files; which explains its inability to correctly classify the files that are smaller in size and lower in complexity. Smaller files that contain defects are likely to be ranked higher by an effort-aware defect prediction model because they require less effort to inspect.

The results are consistent when using another classification technique, spectral clustering, to partition the data and build the local models. To further validate the superiority of the global model, we cluster the data using another unsupervised classification technique (i.e., spectral clustering). We previously noticed that the low performing cluster has a much lower ratio of defective files. We also observed that in the three cases where the local model outperforms the global models, the low performing cluster has the lowest ratios of defective files. Therefore, we use an approach based on spectral clustering, as an unsupervised classification technique, to

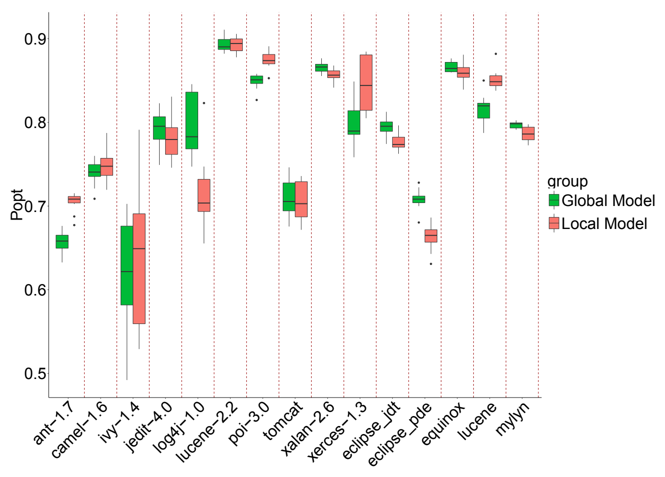


Figure 5: Effort-aware prediction performance (P_{opt}) of global and local models

Table 7: The effect size and difference in performance between global and local models (projects in bold correspond to cases where we observe significant difference between the performances of the two models)

Dataset	Project	P_{opt} (Global)	P_{opt} (Local)	Effect size	p-value
PROMISE	**Ant**	0.73	0.71	large	7.25E-04
	Camel	0.73	0.73	negligible	8.53E-01
	Ivy	0.60	0.63	small	2.79E-01
	Jedit	0.81	0.78	large	2.32E-02
	Log4j	0.79	0.69	large	7.25E-04
	Lucene 2.2	0.92	0.89	large	2.16E-05
	POI	0.89	0.87	large	1.46E-02
	Tomcat	0.68	0.70	large	6.30E-02
	Xalan	0.89	0.85	large	1.08E-05
	Xerces	0.87	0.85	medium	2.17E-01
AEEEM	**Eclipse JDT**	0.79	0.77	large	3.88e-03
	Eclipse PDE	0.70	0.66	large	2.16E-05
	Equinox	0.85	0.85	medium	8.92E-02
	Lucene 2.4	0.81	0.85	large	2.05E-04
	Mylyn	0.79	0.78	large	2.87E-03

Table 8: The effect size and difference in performance between global and local models obtained using spectral clustering

Dataset	Project	P_{opt} (Global)	P_{opt} (Local)	Effect size	p-value
PROMISE	Ant	0.73	0.74	medium	1.05E-01
	Camel	0.73	0.73	negligible	7.96E-01
	Ivy	0.60	0.65	small	2.28E-01
	Jedit	0.81	0.81	small	3.93E-01
	Log4j	0.79	0.78	large	6.30E-02
	Lucene 2.2	0.92	0.89	large	7.57E-05
	POI	0.89	0.88	large	7.52E-02
	Tomcat	0.68	0.72	large	1.50E-03
	Xalan	0.89	0.85	large	1.08E-05
	Xerces	0.87	0.83	large	1.85E-02
AEEEM	**Eclipse JDT**	0.79	0.74	large	6.38E-05
	Eclipse PDE	0.70	0.68	large	1.42E-03
	Equinox	0.85	0.84	large	6.40E-05
	Lucene 2.4	0.81	0.80	large	1.17E-02
	Mylyn	0.79	0.56	large	6.39E-05

separate the training set into a defective and clean clusters. For the spectral clustering based approach, we use the steps proposed by Zhang *et al.* [43] to label the defective cluster (i.e., the cluster with the highest metrics values is labelled as defective). We build the effort-aware prediction model using the defective cluster, and compare its performance with the global effort-aware defect prediction model.

Our results (shown in Table 8) show that the global model is either superior or has similar performance to the local model, even using other classification techniques to partition the data. Since the building of the local models requires some overhead related to obtaining the clusters and assessing which local model is best, it seems to be more practical to train the effort-aware prediction model using the whole dataset.

The global effort-aware defect prediciton model trained on the whole dataset has an overall better performance than the local model. The weakness of the local model lies in its inability to recognize that some of the small files are possibly defective.

4. THREATS TO VALIDITY

This section discusses the threats to validity of our study. We describe all possible threats by following the common guidelines [40].

Threats to conclusion validity concern the relation between the treatment and the outcome. One conclusion validity threat comes from the data clustering method. Using another clustering algorithm might return different conclusions. Therefore, we validate the consistency of the conclusions by testing our approach with different clustering algorithms (e.g., K-means and spectral clustering). We have used non-parametric tests that do not require making assumptions about the distribution of the datasets; thus, not violating assumptions of the constructed statistical models.

Threats to external validity concern the possibility to generalize our results. This study uses lines of code as measure of effort, similir to Mende *et al.* [26] and Bettenburg *et al.* [7]. However, using other measures of efforts such as McCabe cyclomatic complexity to replicate the study can be useful in generalizing the findings about the suitability of locals models for the task of effort-aware defect prediction. Nevertheless, prior work [37] has provided strong evidence of the correlation between lines of code and other measures of efforts, when building effort-aware defect prediction models. Our subject projects are all Java projects. Moreover, verifying our findings on projects with different context factors (e.g., programming languages, application domain, and size) can examine the generalizability of our approach.

Threats to reliability validity concern the possibility of replicating this study. The subject projects are publicly available from [15] and [12]. We attempt to provide the necessary details to replicate our study. We have used off-the-shelf clustering techniques. We also build all defect prediction models using linear regression, a widely used and easily replicated modelling technique.

5. RELATED WORK

5.1 Defect Prediction

Defect prediction models are designed to help improve the quality of software and to reduce the cost of software testing and maintenance. After the creation of the PROMISE repository [28] in 2005 and later the AEEEM dataset [12] in 2010, there has been a growth in the research of defect prediction [39]. Both repositories provide datasets from real-world projects for public use; thus allowing researchers to build comparable and repeatable defect prediction models. Due to the large body of work in the area of defect prediction, we present in this section only some of the studies conducted in this area, with the purpose to highlight the most important aspects of defect prediction.

Defect prediction models are trained on SE data containing actual defect information and a set of descriptive metrics that are believed to reflect defect proneness. To build the defect prediction models, two major types of modelling techniques are used: machine learning methods (e.g., support vector machines, decision trees, and K-nearest neighbour) and statistical methods (e.g., logistic regression and Naive Bayes). Different studies [12][25][2] have evaluated the various modelling techniques for the task of defect prediction. Lessmann *et al.* [25] concluded that the choice of the modelling technique has limited impact on the quality of the defect prediction models. Zhang *et al.* [43] proposed a novel modelling technique based on an unsupervised classifier (i.e., spectral clustering), thus removing the need for a training set and easing the task for cross-project defect prediction.

In addition to the modelling approaches used, the metrics used to describe the software entities have also received much attention by researchers. Some of the most important findings in this regard are that larger and more complex files are more likely to be prone to defects (i.e., product metrics)[13]. Also, heavily changed files with multiple authors have been found to be riskier and likely to contain defects (i.e., process metrics))[31][13][32]. Other findings also suggest that process metrics are more efficient in predicting defect proneness than product metrics [31]

In parallel, research efforts have been deployed to assess the limitations of the defect prediction models. The stability and usefulness of the prediction models have been found to be influenced by the quality of the training data in terms of its representativeness, heterogeneity, and volume [4][8]. The same studies also found that the SE training data does have quality issues and might contain errors [4][8]. However, Kim *et al.*[21] claim that the defect prediction models can still perform reasonably well under the presence of quality issues, such as noise. In this paper, we aim to provide additional insight on defect prediction by confirming some of the prior findings, and reporting our observations on the impact of using different training sets on the quality of the defect prediction.

5.2 Effort-aware Defect Prediction

Traditional prediction models assume that the effort to test and inspect a software entity is the same across all entities. Arisholm *et al.*[1] challenged the traditional prediction models by including a notion of effort into the evaluation. Many of the classification algorithms (e.g., logistic regress, decision trees, and support vector machines) that are good according to classical performance measures performed poorly when assessed with a measure of effort [1][2][3]. Mende *et al.* [26] include the notion of effort in the prediction models and assess the improvement on the cost effectiveness of the model; they conclude that there is a significant improvement of the prediction performance. Kamei *et al.* [16] verify whether some major findings in the defect prediction literature are still valid in effort-aware models. Shihab *et al.* [37] investigate the measures used to assess effort (e.g., lines of code). Effort-aware prediction enables QA teams to select the modules with the highest risk for further treatment [26]. When the effort is measured using lines of code, the risk is equivalent to the defect density. Given the cost-effectiveness of the effort-aware modelling on the available testing resources, it is important to look further into effort-aware prediction models and attempt to improve the performance. We investigate in this work whether local defect prediction models (explained in the subsequent section) can be suitable in the context of effort-aware defect prediction.

5.3 Data Partitioning in Defect Prediction

To improve the performance of prediction models, many strategies are used such as selecting the right metrics to build a model [6][31][13] or predicting defects from different artifacts of a project (e.g., the commit changes [17] or the cached history [22]). However, one of the main problems observed in SE data is its great amount of variability [27]. Therefore, another way to improve the quality of the defect prediction models is to look at partitions of data with similar properties and build models locally rather than globally. This assertion led the way to a line of work comparing global and local models with the purpose to evaluate the performance of local modelling. For instance, studies [27][7] have shown that there exists a benefit in considering models based on clusters of data to decrease the variability in the data and improve the performance. While local models result in a substantially better fit to the underlying data, the improvement to the prediction performance in terms of prediction error has been described as "small" by Bettenburg *et al.*. Moreover, Bettenburg *et al.* find that local models can possibly provide conflicting recommendations to the practitioners. Therefore, they recommend using trends obtained from global models, that take into account local characteristics. In this work, we attempt to replicate the work of prior studies and examine the quality of the individual local models, to gain more insights about the reasons behind the failure and/or success of the local models.

6. CONCLUSIONS

In this study, we start with the evaluation of local and global models for defect prediction and observe that there is always one local model that experiences higher prediction error. Moreover, we find that the cluster associated with the poor performing model has distinguished characteristics including lower ratio of defective files, files with smaller size, lower code complexity and fewer past defects and churn. We exclude the poor performing cluster and train a local effort-aware model with the remaining training set. We observe an overall superiority of the global model that is trained with the whole dataset. By excluding the low performing cluster, the new local model fails to learn from small defective files. The global model, on the other hand, benefits from the variability of the data and is better at predicting the bugginess of such files. Therefore, we do not find that local models bring a worthy improvement over global models for the task of effort-aware modelling, especially given the overhead required to cluster the data before fitting the local models. In the future, special attention should be given to files that are smaller in size when conducting effort-aware prediction. This type of files generally requires less effort, but can easily be misclassified as clean. In the future, it is worth examining whether building global models that take local considerations into account can be beneficial to the task of effort-aware defect prediction, to combine the advantages of both models.

7. REFERENCES

[1] Arisholm, E., and Briand, L. C. Predicting fault-prone components in a java legacy system. In *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering* (New York, NY, USA, 2006), ISESE '06, ACM, pp. 8–17.

[2] Arisholm, E., Briand, L. C., and Fuglerud, M. Data mining techniques for building fault-proneness models in telecom java software. In *Proceedings of the The 18th IEEE International Symposium on Software Reliability* (Washington, DC, USA, 2007), ISSRE '07, IEEE Computer Society, pp. 215–224.

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

[4] Bachmann, A., Bird, C., Rahman, F., Devanbu, P., and Bernstein, A. The missing links: Bugs and bug-fix commits. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering* (New York, NY, USA, 2010), FSE '10, ACM, pp. 97–106.

[5] Basili, V., Briand, L., and Melo, W. A validation of object-oriented design metrics as quality indicators. *Software Engineering, IEEE Transactions on 22*, 10 (Oct 1996), 751–761.

[6] Basili, V. R., Briand, L. C., and Melo, W. L. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng. 22*, 10 (Oct. 1996), 751–761.

[7] Bettenburg, N., Nagappan, M., and Hassan, A. E. Think locally, act globally: Improving defect and effort prediction models. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories* (Piscataway, NJ, USA, 2012), MSR '12, IEEE Press, pp. 60–69.

[8] Bird, C., Bachmann, A., Aune, E., Duffy, J., Bernstein, A., Filkov, V., and Devanbu, P. Fair and balanced?: bias in bug-fix datasets. 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* (2009), ACM, pp. 121–130.

[9] Cliff, N. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin 114*, 3 (Nov. 1993), 494–509.

[10] Cohen. A power primer. Psychological Bulletin, 1992.

[11] Collofello, J. S., and Woodfield, S. N. Evaluating the effectiveness of reliability-assurance techniques. *J. Syst. Softw. 9*, 3 (Mar. 1989), 191–195.

[12] D'Ambros, M., Lanza, M., and Robbes, R. An extensive comparison of bug prediction approaches. In *Proceedings of MSR 2010 (7th IEEE Working Conference on Mining Software Repositories)* (2010), IEEE CS Press, pp. 31 – 41.

[13] Hassan, A. E. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference*

on Software Engineering (Washington, DC, USA, 2009), ICSE '09, IEEE Computer Society, pp. 78–88.

[14] Holm, S. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* (1979), 65–70.

[15] Jureczko, M., and Madeyski, L. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering* (New York, NY, USA, 2010), PROMISE '10, ACM, pp. 9:1–9:10.

[16] Kamei, Y., Matsumoto, S., Monden, A., Matsumoto, K.-i., Adams, B., and Hassan, A. Revisiting common bug prediction findings using effort-aware models. In *Software Maintenance (ICSM), 2010 IEEE International Conference on* (Sept 2010), pp. 1–10.

[17] Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A., and Ubayashi, N. A large-scale empirical study of just-in-time quality assurance. *IEEE Trans. Softw. Eng. 39*, 6 (June 2013), 757–773.

[18] Kaufman, L., and Rousseeuw, P. J. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York, 1990.

[19] Khoshgoftaar, T., Allen, E., Jones, W., and Hudepohl, J. Classification tree models of software quality over multiple releases. In *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on* (1999), pp. 116–125.

[20] Khoshgoftaar, T., and Rebours, P. Improving software quality prediction by noise filtering techniques. *Journal of Computer Science and Technology 22*, 3 (2007), 387–396.

[21] Kim, S., Zhang, H., Wu, R., and Gong, L. Dealing with noise in defect prediction. In *2011 33rd International Conference on Software Engineering (ICSE)* (2011), IEEE, pp. 481–490.

[22] Kim, S., Zimmermann, T., Whitehead Jr., E. J., and Zeller, A. Predicting faults from cached history. In *Proceedings of the 29th International Conference on Software Engineering* (Washington, DC, USA, 2007), ICSE '07, IEEE Computer Society, pp. 489–498.

[23] Kitchenham, B., and Mendes, E. Why comparative effort prediction studies may be invalid. In *Proceedings of the 5th international Conference on Predictor Models in Software Engineering* (2009), ACM, p. 4.

[24] Kruskal, W. H., and Wallis, W. A. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association 47*, 260 (1952), 583–621.

[25] Lessmann, S., Baesens, B., Mues, C., and Pietsch, S. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *Software Engineering, IEEE Transactions on 34*, 4 (July 2008), 485–496.

[26] Mende, T., and Koschke, R. Effort-aware defect prediction models. In *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on* (March 2010), pp. 107–116.

[27] Menzies, T., Butcher, A., Marcus, A., Zimmermann, T., and Cok, D. Local vs. global models for effort estimation and defect prediction. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering* (Washington, DC, USA, 2011), ASE '11, IEEE Computer Society, pp. 343–351.

[28] Menzies, T., Rees-Jones, M., and Krishna, R.and Pape, C. The promise repository of empirical software engineering data, 2015.

[29] Mizuno, O., and Kikuno, T. Prediction of fault-prone software modules using a generic text discriminator. *IEICE - Trans. Inf. Syst. E91-D*, 4 (Apr. 2008), 888–896.

[30] MJ. Bland, D. A. Transformations, means and confidence intervals. *Br Med* (1996).

[31] Moser, R., Pedrycz, W., and Succi, G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of the 30th International Conference on Software Engineering* (New York, NY, USA, 2008), ICSE '08, ACM, pp. 181–190.

[32] Nagappan, N., Ball, T., and Zeller, A. Mining metrics to predict component failures. In *Proceedings of the 28th International Conference on Software Engineering* (New York, NY, USA, 2006), ICSE '06, ACM, pp. 452–461.

[33] Nam, J., Pan, S. J., and Kim, S. Transfer defect learning. In *Proceedings of the 2013 International Conference on Software Engineering* (2013), IEEE Press, pp. 382–391.

[34] Park, H.-S., and Jun, C.-H. A simple and fast algorithm for k-medoids clustering. *Expert Syst. Appl. 36*, 2 (Mar. 2009), 3336–3341.

[35] Ringnér, M. What is principal component analysis? *Nature biotechnology 26*, 3 (2008), 303–304.

[36] Romano, J., Kromrey, J., Coraggio, J., and Skowronek, J. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys? In *annual meeting of the Florida Association of Institutional Research* (2006), pp. 1–3.

[37] Shihab, E., Kamei, Y., Adams, B., and Hassan, A. E. Is lines of code a good measure of effort in effort-aware models? *Information and Software Technology 55*, 11 (Nov. 2013), 1981–1993.

[38] Turney, P. Types of cost in inductive concept learning. pp. 15–21.

[39] Wang, S., and Yao, X. Using class imbalance learning for software defect prediction. *IEEE Transactions on Reliability 62*, 2 (June 2013), 434–443.

[40] Yin, R. K. *Case Study Research: Design and Methods - Third Edition*, 3 ed. SAGE Publications, 2002.

[41] Zar, J. H. *Spearman Rank Correlation*. John Wiley & Sons, Ltd, 2005.

[42] Zhang, F., Mockus, A., Zou, Y., Khomh, F., and Hassan, A. E. How does context affect the distribution of software maintainability metrics? In *Proceedings of the 29th IEEE International Conference on Software Maintainability* (2013), ICSM '13, pp. 350 – 359.

[43] Zhang, F., Zheng, Q., Zou, Y., and Hassan, A. E. Cross-project defect prediction using a connectivity-based unsupervised classifier. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016* (2016), pp. 309–320.

[44] Zimmermann, T., and Nagappan, N. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th International Conference on Software Engineering* (New York, NY, USA, 2008), ICSE '08, ACM, pp. 531–540.