e-Informatica Software Engineering Journal, Volume 9, Issue 1, 2015, pages: 21–35, DOI 10.5277/e-Inf150102

Cross–Project Defect Prediction With Respect To Code Ownership Model: An Empirical Study

Marian Jureczko*, Lech Madeyski**

*Institute of Computer Engineering, Control and Robotics, Wroclaw Univeristy of Technology

**Faculty of Computer Science and Management, Wroclaw University of Technology

marian.jureczko@pwr.edu.pl, lech.madeyski@pwr.edu.pl

Abstract

The paper presents an analysis of 83 versions of industrial, open-source and academic projects. We have empirically evaluated whether those project types constitute separate classes of projects with regard to defect prediction. Statistical tests proved that there exist significant differences between the models trained on the aforementioned project classes. This work makes the next step towards cross-project reusability of defect prediction models and facilitates their adoption, which has been very limited so far.

Keywords: software engineering; defect prediction; empirical study

1. Introduction

Assuring software quality is known to require time-consuming and expensive development processes. The costs generated by those processes may be minimized when the defects are predicted early on, which is possible by means of defect prediction models [1]. Those models, based on software metrics, have been developed by a number of researchers (see Section 2). The software metrics which describe artifacts of the software development process (e.g. software classes, files) are generally used as the models' input. The model output usually estimates the probability of failure, the occurrence of a defect or the expected number of defects. The predictions are made for a given artifact. The idea of building models on the basis of experienced facts, called inductive inference, is discussed in the context of software engineering by Samuelis [2].

Defect prediction models are extraordinarily useful in software testing process. The available resources are usually limited and, therefore, it may be difficult to conduct the comprehensive tests on, and the reviews of, all artifacts. Defect prediction models are extraordinarily useful in software testing process. The available resources are usually limited and, therefore, it may be difficult to conduct the comprehensive tests on, and the reviews of, all artifacts [1]. The predictions may be used to assign varying priorities to different artifacts (e.g. classes) under test [3, 4]. According to the 80:20 empirical rule, a small amount of code (often quantified as 20% of the code) is responsible for the majority of software defects (often quantified as 80% of the known defects in the system) [5, 6]. Therefore, it may be possible to test only a small amount of artifacts and find a large amount of defects. In short, a well-designed defect prediction model may save a lot of testing efforts without decreasing software quality.

The defect prediction models may give substantial benefits, but in order to build a model, a software measurement program must be launched. According to Kaner and Bond [7], only few companies establish such programs, even fewer succeed with them and many of the companies use them only to conform to the criteria laid down in the Capability Maturity Model [8]. The costs may be one of the reasons behind the limited adoption of the defect prediction models, e.g. an average overhead for metrics collection was estimated to be 4–8% of the overall value [8, 9]. Using cross-project defect prediction could reduce the expenditure, since one model may be used in several software projects, which means that it is not necessary to launch a completely new software measurement program for each project. The cross–project defect prediction is also helpful with solving the problems connected with the lack of historical data indispensable to train a model [10]. Unfortunately, the body of knowledge of cross–project defect prediction does not support us with the results that are advanced enough to be used (details in the next section). The intention of this work is to reveal the facts regarding high level prediction boundaries, particularly the possibility of using prediction models across different project code ownership models. The conducted experiments aiming at verifying whether cluster of projects can be derived from the source code ownership model, where the cluster is a group of software projects that share a common prediction model. Such finding eases the application of defect prediction by removing the necessity of training the model. It is enough to identify the cluster a given project belongs to and use the common model.

This study investigates whether there is a relevant difference between industrial, open-source and academic software projects with regard to defect prediction. In order to explore that potential disparity, the data from 24 versions of 5 industrial projects, 42 versions of 13 open-source projects and 17 versions of 17 academic projects were collected. Several defect prediction models were built and the efficiency of the predictions was compared. Statistical methods were used in order to decide whether the obtained differences were significant or not. It is worth mentioning that a somehow related question of suitability of software quality model for projects within different application domains was raised by Villalba et al. [11], whereas the results of other works which investigate the cross-project defect prediction (i.e. [12, 13]) suggest that the code ownership model might be a relevant factor with regard to the prediction performance.

The rest of this paper is organized as follows: subsequent sections describe related work, the design of empirical evaluation used in this study (including the details of data collection and analysis methodology), the descriptive statistics of the collected data, the results of empirical evaluation, threats to the validity of the empirical study, and conclusions.

2. Related Work

Cross-project reusability of defect prediction models would be extremely useful. Such generalized prediction models would serve as a starting point in software development environments that cannot provide historical data and, as a result, they would facilitate the adoption of defect prediction models.

A preliminary work in this area was conducted by Subramanyam and Krishnan [14]. The authors investigated a software project where C++, as well as Java, were employed and found substantial differences between classes written in different programming languages with regard to defect prediction, e.g. the interaction effect (the defect count grows with the CBO value for C++ class but decreases for the Java classes; the relation was calculated with respect to the DIT metric). Hence, the results indicate issues with regard to cross-language predictions. The authors investigated only one project, nevertheless, similar difficulties might arise in cross-project predictions.

Nagappan et al. [15] analyzed whether defect predictors obtained from one project history are applicable to other projects. It turned out that there was no single set of metrics that would fit into all five investigated projects. The defect prediction models, however, could be accurate when obtained from similar projects (the similarity was not precisely defined, though). The study was extended by Zimmerman et al. [12], who performed 622 cross-project predictions for 12 real world applications. A project was considered as a strong predictor for another project when all precision, recall, and accuracy were greater than 0.75. Only 21 cross-project validations satisfied this criterion, which sets the success rate at 3.4%. Subsequently, the guidelines to assess the cross-project prediction chance of success were given. The guidelines were summarized in a decision tree. The authors constructed separate trees for assessing prediction precision, recall, and accuracy, but only the tree for precision was given in the paper.

A study of cross-company defect prediction was conducted by Turhan et al. [10]. The authors concluded that there is no single set of static code features (metrics) that may serve as defect predictor for all the software projects. The effectiveness of the defect prediction models was measured using probability of detection (pd) and probability of false alarm (pf). Cross-company defect prediction dramatically increased pd as well as pf. The authors were also able to decrease the pf by applying the nearest neighbor filtering. The similarity measure was the Euclidean distance between the static code features. However, there was still a drawback for cross-company defect prediction models: the project features which might influence the effectiveness of cross-company predictions were not identified.

In an earlier paper [13], we presented an empirical study showing that data mining techniques may be used to identify project clusters with regard to cross-project defect prediction. The k-means and Kohonen's neural networks were applied to correlation vectors in order to identify the clusters. The correlation vectors were calculated for each version of each project respectively and represented Pearson's correlation coefficients between software metrics and numbers of defects. Subsequently, a defect prediction model was created for each identified cluster. In order to validate the existence of a cluster, the efficiency of the cluster model was compared with the efficiency of a general model. The general model was trained using data from all the projects. Six different clusters were identified and the existence of two of them was statistically proven. The clusters characteristics were consistent with Zimmerman's findings [12] about the factors that are critical in cross-project prediction. In our paper, we make a step towards simplifying the setup of defect prediction in the software development process. The laborious activities regarding calculation of correlation vectors and mining the clusters are not needed as it is obvious from the very beginning to which cluster a project belongs. A subset of the same data set had already been analysed in [16,17]. In [16] 5 industrial and 11 open–source projects were investigated. The study was focused on the role of the size factor in defect prediction. The other paper was focused on the cross-project defect prediction. In [17], published in Polish, being a preliminary study to this one, we focused on the differences and similarities between industrial, open–source and academic projects, whereas in this paper, we additionally performed a comprehensive statistical analysis. As a result, new projects may take advantage of the prediction models developed for the aforementioned classes of the existing projects.

It is also worth mentioning that a different approach, based on the idea of inclusion additional software projects during the training process, can provide a cross-project perspective on software quality modelling and prediction [18].

A comprehensive study of cross-project defect prediction was conducted by He et al. [19]. The authors investigated 10 open source projects to check whether training data from other projects can provide better prediction results than training data from the same project - in the best cases it was possible. Furthermore, in 18 out of 34 cases the authors were able to obtain a Recall greater than 70% and a Precision greater than 50% for the cross-project defect prediction.

3. Empirical Evaluation Design

3.1. Data Collection

The data from 83 versions of 35 projects was collected and analyzed. It covers 24 versions of 5 industrial projects, 42 versions of 13 open–source projects and 17 versions of 17 academic projects.

The number of versions is greater than the number of projects because there were projects in which data from several versions were collected. For example, in the case of Apache Ant project (http://ant.apache.org), versions 1.3, 1.4, 1.5, 1.6 and 1.7 were analyzed. For each of the analysed versions there was an external release, which was visible to the customer or user.

Each of the investigated industrial projects is a custom-built enterprise solution. All of the industrial projects have been already successfully developed by different teams of 10 to 40 developers and installed in the customer environments. All of them belong to the insurance domain but implement different feature sets on top of Java-based frameworks. Each of the industrial projects were developed by the same vendor.

The following open-source projects were investigated: Apache Ant, Apache Camel, Apache Forrest, Apache Log4j, Apache Lucene, Apache POI, Apache Synapse, Apache Tomcat, Apache Velocity, Apache Xalan, Apache Xerces, Ckjm and Pbeans.

The academic projects were developed by the fourth and the fifth-year graduate MSc computer science students. The students were divided into the groups of 3, 4 or 5 persons. Each group developed exactly one project. The development process was highly iterative (feature driven development). Each project lasted one year. During the development for each feature UML documentation was prepared. Furthermore, high level of test code coverage was obtained by using latest testing tools, e.g. JUnit for unit tests, FitNesse for functional tests. The last month of development was used for additional quality assurance and bug fixing; the quality assurance was conducted by external group of subjects (i.e. not the subjects involved in the development). The projects were from different domains, were built using different frameworks, had different architectures and covered different sets of functionalities, nonetheless all of them were written in Java.

The objects of the measurement were software development products (Java classes). The following software metrics were used in the study:

- Chidamber & Kemerer metrics suite [20]: Weighted Method per Class (WMC), Depth of Inheritance Tree (DIT), Number Of Children (NOC), Coupling Between Object classes (CBO), Response For a Class (RFC) and Lack of Cohesion in Methods (LCOM);
- a metric suggested by Henderson-Sellers [21]: Lack of Cohesion in Methods (LCOM3);
- Martin's metrics [22]: Afferent Couplings (Ca) and Efferent Couplings (Ce).
- QMOOD metrics suite [23]: Number of Public Methods (NPM), Data Access Metric (DAM), Measure Of Aggregation (MOA), Measure of Functional Abstraction (MFA) and Cohesion Among Methods (CAM);
- quality oriented extension of Chidamber & Kemerer metrics suite [24]: Inheritance Coupling (IC), Coupling Between Methods (CBM) and Average Method Complexity (AMC),
- two metrics, that are based on the McCabe's cyclomatic complexity measure [25]: Maximal Cyclomatic Complexity (Max_CC) and Average Cyclomatic Complexity (Avg_CC),
- Lines Of Code (LOC),
- Defects — the dependent variable; in the case of industrial and open source projects the sources of the defect data were testers and end users, in the case of academic projects the source of the defect data were students (not involved in a development of a particular projects) who validated the developed software against the specification during the last month (devoted to testing) of the 1-year project.

Definitions of the metrics listed above can be found in [16]. In order to collect the metrics, we used a tool called Ckjm (http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm). The version of Ckjm employed here was reported earlier by Jureczko and Spinellis [16]). The defect count was collected with a tool called BugInfo (http://kenai.com/projects/buginfo). The collected metrics are available online in a Metric Repository (all metrics: http://purl.org/MarianJureczko/MetricsRepo, metrics used in this research: http://purl.org/MarianJureczko/MetricsRepo/IET_CrossProjectPrediction). Data sets related to the analyzed software defect prediction models are available from the R package [26] to streamline reproducible research [27, 28].

The employed metrics might be considered as the classic ones. Each of them is in use for at least several years. Hence, the metrics are well known, already recognized by the industry and have a good tool support. There is a number of other metrics, some of them very promising in the field of defect prediction, e.g. the cognitive, the dynamic and the historical metrics [15, 29, 30]. A promising set of metrics are process metrics analysed by Madeyski and Jureczko [31]. Some of them, like Number of Distinct Committers (NDC) or Number of Modified Lines (NML), can significantly improve defect prediction models based on classic software product metrics [31]. Nevertheless, we decided not to use them, since those metrics are not so popular yet (especially in the industry) and the collecting process could be challenging. One may expect that the limited tool support would result in decreasing the number of investigated projects, which is a crucial factor in cross-project defect prediction. We fully understand and accept the value of using metrics that describe a variety of features. Furthermore, we are involved in the development of a tool which includes support for the historical metrics [32]. We are going to use the tool to collect metrics for future experiments.

3.2. Data Analysis Methodology

The empirical evaluation described further was performed to verify whether there is a difference between industrial, open–source and academic projects with regard to defect prediction. Statistical hypotheses were formulated. The defect prediction models were built and applied to the investigated projects. The efficiency of prediction was used to evaluate the models and to verify the hypotheses.

To render that in a formal way, it is necessary to assume that $E(M, v)$ is the evaluation function. The function assesses the efficiency of prediction of the model M on the version v of the investigated software project. Let $c_1, c_2, ..., c_n$ be the classes from the version v in descending order of predicted defects according to the model M, and let $d_1, d_2, ..., d_n$ be the number of defects in each class. D_i is the $\sum(d_1, ..., d_i)$, i.e., the total defects in the first i classes. Let k be the smallest index so that $D_k > 0.8 * D_n$, then $E(M, v) = k/n * 100\%$. Such evaluation function has been used as it clearly corresponds with the software projects reality we faced with. It is closely related to a quality goal: detect at least 80% of defects. The evaluation function shows how many classes must be tested (in practice it corresponds well to how much effort must be committed) to reach the goal when testing according to prediction model output. The properties of the group of evaluation functions that the one selected by us belongs to have been analysed by Weyuker et al. [33].

The empirical evaluation is defined in a generic way which embraces three investigated classes of software projects. Let A, B and C be those classes (i.e. industrial, open–source and academic, respectively). Let us interpret the classes of software projects as the sets of versions of software projects. Let A be the object of the current empirical evaluation. B and C will be investigated in subsequent experiments using the analogous procedure. Let a be a member of the set A, $a \in A$. Let M_x be the defect prediction model which was trained using data from versions that belong to set X. Specifically, there are M_A, M_B, M_C and $M_{B \cup C}(B \cap C = \neg A)$. Subsequently, the following values were calculated for each $a \in A$:

- $E(M_A, a)$ – let's call the set of obtained values E_A,
- $E(M_B, a)$ – let's call the set of obtained values E_B,
- $E(M_C, a)$ – lets call the set of obtained values E_C,
- $E(M_{B \cup C}, a)$ – let's call the set of obtained values $E_{B \cup C}$.

The following statistical hypothesis may be formulated with the use of E_A, E_B, E_C and $E_{B \cup C}$ sets:

- $H_{0,A,B}$ – E_A and E_B come from the same distribution.
- $H_{0,A,C}$ – E_A and E_C come from the same distribution.

Table 1. Descriptive statistics of metrics in different projects classes ($\overline{X}$ – mean; s – standard deviation; r – Pearson correlation coefficient; $*$ – correlation significant at 0,05 level)

	Industrial			Open–source			Academic		
	$\overline{X}$	s	r	$\overline{X}$	s	r	$\overline{X}$	s	r
WMC	5.2	8.6	0.13∗	10.5	13.7	0.29∗	9.7	10.8	0.38∗
DIT	3	1.7	−0.07∗	2.1	1.3	−0.01	2.1	1.7	0.29∗
NOC	0.6	8.8	0	0.5	3.4	0.03∗	0.2	1.5	−0.04
CBO	15.1	20.8	0.26∗	10.3	16.9	0.20∗	8	8.1	0.25∗
RFC	24.7	27.5	0.28∗	27.1	33.3	0.34∗	26	30.6	0.53∗
LCOM	37.6	660.2	0.03∗	95.2	532.9	0.19∗	62	276.5	0.37∗
Ca	2.8	17.7	0.16∗	5.1	15.3	0.11∗	3.8	6.7	0.19∗
Ce	12.3	10.3	0.24∗	5.4	7.4	0.26∗	4.7	5.9	0.38∗
NPM	3.4	7.9	0.08∗	8.4	11.5	0.22∗	7.4	8.7	0.08∗
LCOM3	1.4	0.6	−0.04∗	1.1	0.7	−0.07∗	1.1	0.6	−0.11∗
LOC	170.4	366.7	0.26∗	281.3	614.7	0.29∗	248.1	463.4	0.53∗
DAM	0.2	0.3	0.02∗	0.5	0.5	0.06∗	0.6	0.5	0.07∗
MOA	0.1	1	0.03∗	0.8	1.8	0.27∗	0.8	1.6	0.27∗
MFA	0.6	0.4	−0.06∗	0.4	0.4	−0.02∗	0.3	0.4	0.16∗
CAM	0.6	0.2	−0.13∗	0.5	0.3	−0.19∗	0.5	0.2	−0.17∗
IC	1.1	1.1	−0.04∗	0.5	0.8	0.06∗	0.3	0.5	−0.03
CBM	1.7	2.4	−0.03∗	1.5	3.1	0.10∗	0.5	1.5	0.02
AMC	30.4	39.8	0.14∗	28.1	80.7	0.07∗	21.2	24.7	0.24∗
Max_cc	3.3	5.7	0.17∗	3.8	7.5	0.17∗	3.2	5.5	0.31∗
Avg_cc	1.3	1.5	0.13∗	1.3	1.1	0.12∗	1.1	0.8	0.17
Defects	0.232	0.887		0.738	1.906		0.382	1.013	

- $H_{0,A,B\cup C}$ – E_A and $E_{B\cup C}$ come from the same distribution.

The alternative hypothesis:

- $H_{1,A,B}$ – E_A and E_B come from different distributions.
- $H_{1,A,C}$ – E_A and E_C come from different distributions.
- $H_{1,A,B\cup C}$ – E_A and $E_{B\cup C}$ come from different distributions.

When the alternative hypothesis is accepted and mean(EA) < mean(EX), there is a significant difference in prediction accuracy: the model trained on the data from set A gives a significantly better prediction than the model trained on the data from set X. The predictions are made for all the project versions which belong to the set A. Hence, the data from set X should not be used to build defect prediction models for the project versions which belong to set A.

The hypotheses were evaluated by the parametric t-test for dependent samples. The general assumptions of parametric tests were investigated beforehand. The homogeneity of variance was tested using Levene's test and the normality of distribution was tested using the Shapiro-Wilk test [34]. All hypotheses were tested on the default significance level: $\alpha = 0.05$.

4. Experiments and Results

4.1. Descriptive Statistics

The three aforementioned classes of software projects, namely: industrial, open–source and academic, were described in Table 1. The description provides information about the mean value and standard deviation of each of the analyzed software metrics. Each metric is calculated per Java class, and the statistics are based on all Java classes in all projets that belong to a given class of projects. Moreover, the correlations with the number of defects were calculated and presented. Most of the size related metrics (WMC, LCOM, LOC, Max_cc and Avg_cc) had higher values in open-source projects, while coupling metrics (CBO and Ce) had the greatest values in the industrial projects. In the case of eachof the in-

Table 2. The number of classes per project

	Industrial		Open–source		Academic	
	$\overline{X}$	s	$\overline{X}$	s	$\overline{X}$	s
No of classes	2806.3	831.7	302.6	219.4	56.4	58.4

Table 3. The number of defects per Java class

Type	$\overline{X}$	s
Open–source	.7384	1.9
Industrial	.2323	.9
Academic	.3816	1.0

vestigated project classes, the RFC metric has a higher correlation coefficient with the number of defects. However, there are major differences in its value (it is 0.28 in the case of the industrial projects, and 0.53 in the case of the academic projects) and in the sets of other metrics that are highly correlated with the number of defects.

Collected data suggest that in all kinds of the projects (industrial, open–source and academic) the mean LOC per class follows the rule of thumb presented by Kan in his book [35] (Table 12.2 in Chapter 12) that LOC per C++ class should be less than 480. The similar value is expected for Java. Bigger classes would suggest poor object-oriented design. Low LOC per class does not mean that the code under examination is small. To give an impression with regard to the size of the investigated projects Table 2 presents numbers of classes per project.

The number of defects per Java class is presented in Table 3. It appears that the number of defects per Java class in industrial and academic projects is close with regards to standard deviation and mean, while open source projects are characterized by higher standard deviation as well as mean. The plausible explanation is that lower standard deviations in academic and industrial projects come from more homogeneous development environment than in open source projects.

4.2. Empirical Evaluation Results

The empirical evaluation has been conducted three times. Each time a different class of software projects was used as the object of study.

4.2.1. Industrial Projects

The descriptive statistics are summarized in Table 4. The models that were trained on the data from the industrial projects ('Industrial models' in Table 4) gave the most accurate prediction. The predictions were made only for the industrial projects. Furthermore, the mean value of the evaluation function $E(M, v)$ was equal to 50.82 in the case of the 'Industrial models'. In the case of the other models, the mean values equaled 53.96, 55.38 and 73.59. A smaller value of the evaluation function implies better predictions.

The predictions obtained from the models trained on the data from the industrial projects were compared with the predictions from the other models. The predictions were made only for the industrial projects. The difference was statistically significant only in the case of comparison with the predictions from models trained on the data from the academic projects (see Table 5). In this case the calculated effect size $d = 1.56$ is extremely high (according to magnitude labels proposed by Cohen d effect size equal to .2 is considered small, equal to .5 is considered medium, while equal to .8 is considered high) while the power of a test (i.e., the probability of rejecting H_0 when in fact it is false) is equal to 1. This important finding shows that there exist significant differences between industrial and academic software projects with respect to defect prediction.

4.2.2. Open–source Projects

The descriptive statistics of the results of applying defect prediction models to open–source projects are presented in Table 6. The models, which were trained on the data from the open–source projects ('Open–source models' in Table 6), gave the most accurate prediction.

Table 7 shows the t–test statistics, power and effect size calculations for the open-source projects.

In all of the cases p-value is lower than .05. However, instead of coming to the conclusions

Table 4. Models evaluations for the industrial projects

Model	Non–industrial	Open–source	Academic	Industrial
$\overline{X}$	53.96	55.38	73.59	50.82
s	13.29	12.01	9.68	9.86

Table 5. Dependent samples t-test for the industrial projects.

	$H_{0,\text{ind},\text{open}\cup\text{acad}}$	$H_{0,\text{ind},\text{open}}$	$H_{0,\text{ind},\text{acad}}$
$t, df = 23$	$-.979$	-1.482	-7.637
p	.338	.152	.000
effect size d	.200	.302	1.559
power	.244	.418	1

Table 6. Models evaluations for the open–source projects.

Model	Industrial	Non-open–source	Academic	Open–source
$\overline{X}$	57.67	57.26	65.17	54.00
s	19.22	18.02	14.31	16.66

now we suggest to perform the Bonferroni correction (explained in Section 4.2.4) beforehand. Calculated effect sizes are between medium and small int the first two cases, while medium to large in the last case which is an important finding suggesting serious differences between open source and academic projects with respect to defect prediction. The power of a test is calculated as well suggesting very high (close to 1) probability of rejecting H_0 when in fact it is false.

4.2.3. Academic Projects

The descriptive statistics of the results of applying defect prediction models to academic projects are presented in Table 8. The obtained results are surprising. The 'Academic models' gave almost the worst predictions. Slightly worse were only the 'Industrial models', whilst the 'Open–source models' and 'Non–academic models' gave definitely better predictions.

The analysis presented in Table 9 is based on the t–test for dependent samples. The predictions obtained from the models that were trained on the data from the academic projects were compared with predictions from the other models. The predictions were made only for the academic projects. The differences were not statistically significant, the effect sizes was below small (in the first and third case) and between small and medium in the second case.

4.2.4. Bonferroni Correction

Since several different hypotheses were tested on the same data set, the following kinds of errors were likely to occur: errors in inference, including confidence intervals which fail to include their corresponding population parameters, or hypothesis tests that incorrectly reject the null hypothesis. Among several statistical techniques that have been developed to prevent such instances is the Bonferroni correction. The correction is based on the idea that if n dependent or independent hypotheses are being tested on a data set, then one way of maintaining the familywise error rate is to test each individual hypothesis at a statistical significance level of $1/n$ times. In our case $n = 9$ as there are nine different hypotheses. Hence, the significance level should be decreased to $0.05/9 = 0.0055$. Consequently, the $H_{0,\text{ind},\text{acad}}$ and $H_{0,\text{open},\text{acad}}$ hypotheses will be rejected but $H_{0,\text{open},\text{ind}}$ and $H_{0,\text{open},\text{ind}\cup\text{acad}}$ hypotheses will not be rejected.

Table 7. Dependent samples t–test for the open–source projects.

	$H_{0,\text{open,ind}}$	$H_{0,\text{open,ind}\cup\text{acad}}$	$H_{0,\text{open,acad}}$
$t, df = 41$	−2.363	−2.193	−4.325
p	.023	.034	.000
effect size d	.365	.338	.667
power	.752	.695	.995

Table 8. Models evaluations for the academic projects.

Model	Industrial	Open–source	Non–academic	Academic
$\overline{X}$	56.34	50.60	53.19	55.02
s	20.71	15.56	18.54	20.21

Table 9. Dependent samples t–test for the academic projects.

	$H_{0,\text{acad,ind}}$	$H_{0,\text{acad,open}}$	$H_{0,\text{acad,ind}\cup\text{open}}$
$t, df = 16$	0.312	−1.484	−0.696
p	.759	.157	.496
effect size d	.076	.360	.169
power	.009	.412	.164

4.3. Which Metrics are Relevant?

There are some evidences that the three investigated code ownership models differs with respect to defect prediction. Therefore, it could be helpful for further research to identify the casual relations that drive those differences. Unfortunately, the scope of software metrics (independent variables of the defect prediction models) does not cover many aspects that may be the direct cause of defects. Let us assume that there is an inexperienced developer that gets a requirement to implement and he does something wrong, he introduces a defect into the system. The true cause of the defect is a composition of several factors including the developer experience, requirement complexity and the level of maintainability of the parts of the system that were changed by the developer. Only a fraction of the aforementioned factors can be covered by the metrics available from software repositories and hence many of them must be ignored by the defect prediction models. Taking into consideration the above arguments we decided not to define the casual relations upfront, but investigate what emerges from the models we obtained. Since it does not follow the commonly used procedure (start with a theory and then look for confirmation in empirical data), it is important to keep in mind that results of such analysis does not indicate casual relations, but only coexistence of some phenomena.

The analysis is based on the relevancy of particular metrics in models obtained for different code ownership types. The defect prediction model has the following form:

$$\mathit{ExpectedNumberOfDefects} = a_1 * M_1 + a_2 * M_2 \ldots$$

where a_i represents coefficients obtained from regression, M_i software metrics (independent variables in the prediction). For each metric we calculated its importance factor (the factors are calculated for each code ownership model respectively) using the following form:

$$IF_{Mi} = \frac{a_i * \overline{M_i}}{\sum_j |a_j * \overline{M_j}|}$$

where $\overline{M_i}$ is the average value of metric M_i in the type of code ownership for which the factor is calculated (the averages are reported in Tab. 1). The above definition results in a factor that shows for

Model	Industrial	Open–source	Academic
WMC	−0.06	0	0
DIT	−0.08	0	0.08
NOC	−0.01	0	0
CBO	0.17	0	−0.36
RFC	0.19	0.35	0.04
LCOM	0	0	0
Ca	0	0.05	0.17
Ce	0	0.17	0.20
NPM	0	0	0.01
LCOM3	0.13	0	−0.05
LOC	0.05	0.14	0
DAM	−0.01	−0.17	−0.04
MOA	0	0.12	0
MFA	0.05	0	−0.02
CAM	−0.16	0	0
IC	0	0	−0.02
CBM	−0.03	0	0.01
AMC	−0.04	0	0
Max_cc	0.01	0	0
Avg_cc	0	0	0

Table 10. Importance of metrics in different code ownership models.

given metric what part of the prediction model output is driven by the metric and additionally preserves the sign of the metric's contribution. The obtained values of importance factors are reported in Tab. 10.

The importance factors show that there are similarities as well as differences between the prediction models trained for different types of code ownership. In all of them an important role is played by the RFC metric and all of them take into consideration coupling related metrics. However, in the case of academic projects these are Ca and Ce, in the case of open–source Ce is much more important than Ca and for industrial projects only CBO matters. There are also metrics with positive contribution in only one type of projects (positive contribution means that the metric value grows with the number of expected defects), i.e. LCOM3 for industrial projects (as well as the mentioned earlier CBO), MOA and LOC for the open–source projects, DIT for the academic ones. More significant differences have been observed with regard to negative contribution. The greatest negative contribution for academic projects have CBO and LCOM3, for open–source DAM and for industrial CAM.

5. Threats to Validity

5.1. Construct Validity

Threats to construct validity refer to the extent to which the measures accurately reflect the theoretical concepts they are intended to measure [34]. The mono-method bias reflects the risk of a single means of recording measures. As a result, an important construct validity threat is that we cannot guarantee that all the links between bugs and versioning system files, and, subsequently, classes, are retrieved (e.g. when there is no bug reference identifier in the commit comment), as bugs are identified according to the comments in the source code version control system. In fact, this is a widely known problem and the method that we adopted is not only broadly used but also represents the state of the art with respect to linking bugs to versioning system files and classes [36, 37].

A closely related threat concerns anonymous inner classes. We cannot distinguish whether a bug is related to anonymous inner classes or their containing class, due to the file-based nature of source code version control systems. Hence, it is a common practice not to take into consideration the inner classes [38–40]. Fortunately, the inner classes usually constitute a small portion of all classes (in our study it was 8.84%).

Furthermore, the guidelines of commenting bugfixes may vary among different projects. Therefore, it is possible that the interpretation of the term bug is not unique among the investigated projects. Antoniol et al. [41] showed that a fraction of issues marked as bugs are problems unrelated to corrective maintenance. We did our best to remove such occurrences manually but in future research we plan to apply the suggestion by Antoniol et al. to filter the non-bug issues out.

It is also worth mentioning that it was not possible to track operations like changing the class name or moving the class between packages. Therefore, after such a change, the class is interpreted as a new one.

5.2. Statistical Conclusion Validity

Threats to statistical conclusion validity relate to the issues that affect the validity of inferences. In our study we used robust statistical tools: SPSS and Statistica.

5.3. Internal Validity

The threats to internal validity concern the true causes (e.g., external factors) that may affect the outcomes observed in the study. The external factor we are aware of is the human factor pointed out by D'Ambros et al. [38]. D'Ambros et al. decided to limit the human factor as far as possible and chose not to consider bug severity as a weight factor when evaluating the number of defects. We decided to follow this approach, as Ostrand et al. reported how those severity ratings are highly subjective and inaccurate [42].

Unfortunatley, each of the investigated clusters (i.e. code ownership models) has limited variability. All academic projects were developed at the same university. However, they differ a lot with respect to requirements and architecture. Only two open–source projects (PBeans and ckjm) do not come from Apache and all of them can be classified as a tool or library what is in opposition to industrial projects which are enterprise solutions that employ database systems. Furthermore, all the industrial projects were developed by the same vendor which poses a major threat to external validity. In consequence, it is possible to define an alternative hypothesis that explains the differences between clusters, e.g. the open–source cluster can be redefined into tools & libraries, while the industrial cluster can be redefined into enterprise database oriented solution and then a hypothesis that regards difference between such clusters may be formulated. Unfortunately, those alternative hypotheses cannot be invalidated without additional projects and even with additional projects we cannot avoid further alternative hypotheses with more fancy definitions of cluster boundaries. The root cause of this issue is the sample selection procedure which does not guarantee random selection. Only small part of the population of software projects is available for researchers and collecting data for an experiment is a huge challenge taking into account how difficult is to get access to the source code or software metrics of real, industrial projects. We were addressing this issue by taking into consideration the greatest possible number of projects we were able to cover with a common set of software metrics. It does not solve the issue, but reduces the risk of accepting wrong hypothesis due to some data constellations that are a consequence of sample selection.

5.4. External Validity

The threats to external validity refer to the generalization of research findings. Fortunately, in our study we considered a wide range of different kinds of projects. They represent different ownership models (industrial, open source and academic), belong to different application domains and have been developed according to different software development processes. However, our selection of projects is by no means representative and that poses a major threat to external validity. For example, we only considered software projects developed in Java programming language. Fortunately, thanks to this limitation all the code metrics are defined identically for each system, so we have alleviated the parsing bias.

6. Conclusions

Our study has compared three classes of software projects (industrial, open–source and academic) with regard to defect prediction. The analysis comprised the data collected from 24 versions of 5 industrial projects, 42 versions of 13 open–source projects, and 17 versions of 17 academic projects. In order to identify differences among the classes of software projects listed above, defect prediction models were created and applied. Each of the software project classes was investigated through verifying three statistical hypotheses. The following two noteworthy findings were identified: two of the investigated hypotheses were rejected: $H_{0,ind,acad}$ and $H_{0,open,acad}$. In the case of $H_{0,open,ind \cup acad}$ and $H_{0,open,ind}$ p-values were below .05, but the hypotheses can not be rejected due to the Bonferroni correction. Such results are not conclusive due to threats to external validity discussed in Secton 5.4, as well as the possibility that even small changes in the input data may change the decision regarding hypothesis rejection in both directions and thus we encourage further investigation.

As a result, we obtained some evidence that the open–source, industrial and academic projects may be treated as separate classes of projects with regard to defects prediction. In consequence, we recommend against using models trained on the projects from different code ownership model, e.g. making predictions for an industrial project with a model trained on academic or open–source projects. Of course the investigated classes (i.e. academic, industrial and open-source) may not be optimal and smaller classes could be identified in order to increase the prediction performance. Identification of smaller projects classes constitutes a promising direction for further research.

The prediction models trained for each of the investigated classes of projects were further analysed in order to reveal key differences between them. Let us focus on the differences between open–sources and industrial ones as we have very limited evidences to support the thesis that academic projects constitutes a solid class of projects with respect to defect prediction. However the analysis revealed an almost self–explaining fact with regard to this class of projects. Namely, the deeper the inheritance tree the more likely a student will introduce a defect in such a class. Typically, API of a class with a number of ancestors is spread among the parent classes and thus a developer that looks at only some of them may not understand the overall concept and introduce changes that are in conflict with the source code of one of the other classes. In consequence, we may expect that inexperienced developer, e.g. a student, will miss some important details.

The differences between open–source and industrial projects can be explained by the so called crowd–driven software development model commonly used in open–source projects. High value of the model output in those projects is mainly driven by the LOC and MOA metrics. First of them simply represents size of a class and it is not surprising that it could be challenging to understand a big class for someone who commits to a project only occasionally. Furthermore, the MOA metric can be considered in terms of number of classes that must be known and understood to effectively work with a given one, which creates additional challenges for developers that do not work in a project in a daily manner. There also is the negative contribution of the DAM metrics which also fits well to the picture as high values of this metric correspond with low number of public attributes and thus narrows the scope of source code that a developer should be familiar with. Both, industrial and open–source models use the RFC metric, however, in the case of open–source its more relevant which also supports the aforementioned hypothesis regarding crowd-driven development. The industrial projects are usually developed by people who know the project under development very well. That does not mean that everyone know everything about each class. When it is necessary to be familiar with a number of different classes to make a single change in the project it is still likely to introduce a defect. However, in the case of industrial projects the effect is not so strong. Furthermore, the industrial prediction model uses metrics that regard flaws in the class internal structure, i.e. LCOM3 and

CAM), which make challenges in the development regardless of developer knowledge about other classes.

The models that were trained on the academic projects gave usually the worst predictions. Even in the case of making predictions for academic projects, the models trained on the academic projects did not perform well. It was not the primary goal of the study, but the obtained results made it possible to arrive at that newsworthy conclusion. The academic projects are not good as a training set for the defect prediction models. Probably, they are too immature and thus have too chaotic structure. The obtained results point to the need of reconsidering the relevancy of the studies on defect prediction that rely solely on the data from the academic projects. Academic data sets were used even in the frequently cited [43–45] and recently conducted [46, 47] studies.

Detailed data are available via a web–based metrics repository established by the authors (http://purl.org/MarianJureczko/MetricsRepo). The collected data may be used by other researchers for replicating presented here experiments as well as conducting own empirical studies. The obtained defect prediction models related to the conducted empirical study presented in this paper are available online (http://purl.org/MarianJureczko/IET_CrossProjectPrediction). However, we recommend using the models for cross–project defect prediction with great caution, since the obtained prediction performance is moderate and presumable in most cases can be surpassed by a project specific model.

References

[1] L. Briand, W. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, 2002, pp. 706–720.

[2] L. Samuelis, "On principles of software engineering-role of the inductive inference," *e-Informatica Software Engineering Journal*, Vol. 6, No. 1, 2012, pp. 71–77.

[3] L. Fernandez, P. J. Lara, and J. J. Cuadrado, "Efficient software quality assurance approaches oriented to UML models in real life," *Idea Group Pulishing*, 2007, pp. 385–426.

[4] M. L. Hutcheson and L. Marnie, *Software testing fundamentals*. John Wiley & Sons, 2003.

[5] B. W. Boehm, "Understanding and controlling software costs," *Journal of Parametrics*, Vol. 8, No. 1, 1988, pp. 32–68.

[6] G. Denaro and M. Pezzè, "An empirical evaluation of fault-proneness models," in *Proceedings of the 24rd International Conference on Software Engineering, 2002. ICSE 2002.* IEEE, 2002, pp. 241–251.

[7] C. Kaner and W. P. Bond, "Software engineering metrics: What do they measure and how do we know?" in *10th International Software Metrics Symposium.* IEEE, 2004, p. 6.

[8] N. E. Fenton and M. Neil, "Software metrics: successes, failures and new directions," *Journal of Systems and Software*, Vol. 47, No. 2, 1999, pp. 149–157.

[9] T. Hall and N. Fenton, "Implementing effective software metrics programs," *IEEE Software*, Vol. 14, No. 2, 1997, pp. 55–65.

[10] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, Vol. 14, No. 5, 2009, pp. 540–578.

[11] M. T. Villalba, L. Fernández-Sanz, and J. Martínez, "Empirical support for the generation of domain-oriented quality models," *IET software*, Vol. 4, No. 1, 2010, pp. 1–14.

[12] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy, "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, 2009, pp. 91–100.

[13] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering.* ACM, 2010, p. 9.

[14] R. Subramanyam and M. S. Krishnan, *IEEE Transactions on Software Engineering*, Vol. 29, No. 4, 2003, pp. 297–310.

[15] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceedings of the 28th international conference on Software engineering.* ACM, 2006, pp. 452–461.

[16] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," in *Models and Methods of System Dependability.* Oficyna Wydawnicza

Politechniki Wrocławskiej, 2010, pp. 69–81.

[17] M. Jureczko and L. Madeyski, "Predykcja defektów na podstawie metryk oprogramowania – identyfikacja klas projektów," in *Proceedings of the Krajowa Konferencja Inżynierii Oprogramowania (KKIO 2010)*. PWNT, 2010, pp. 185–192.

[18] Y. Liu, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary optimization of software quality modeling with multiple repositories," *Software Engineering, IEEE Transactions on*, Vol. 36, No. 6, 2010, pp. 852–864.

[19] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, Vol. 19, No. 2, 2012, pp. 167–199.

[20] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, 1994, pp. 476–493.

[21] B. H. Sellers, *Object-Oriented Metrics. Measures of Complexity*. Prentice Hall, 1996.

[22] R. Martin, "OO design quality metrics," *An analysis of dependencies*, 1994.

[23] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering*, Vol. 28, No. 1, 2002, pp. 4–17.

[24] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," in *Sixth International Software Metrics Symposium, 1999. Proceedings*. IEEE, 1999, pp. 242–249.

[25] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, No. 4, 1976, pp. 308–320.

[26] L. Madeyski, *reproducer: Reproduce Statistical Analyses and Meta-Analyses*, 2015, R package. [Online]. http://CRAN.R-project.org//package=reproducer

[27] L. Madeyski and B. A. Kitchenham, "Reproducible Research – What, Why and How," Wroclaw University of Technology, PRE W08/2015/P-020, 2015.

[28] L. Madeyski, B. A. Kitchenham, and S. L. Pfleeger, "Why Reproducible Research is Beneficial for Security Research," *(under review)*, 2015.

[29] J. K. Chhabra and V. Gupta, "A survey of dynamic software metrics," *Journal of computer science and technology*, Vol. 25, No. 5, 2010, pp. 1016–1029.

[30] S. Misra, M. Koyuncu, M. Crasso, C. Mateos, and A. Zunino, "A suite of cognitive complexity metrics," in *Computational Science and Its Applications–ICCSA 2012*. Springer, 2012, pp. 234–247.

[31] L. Madeyski and M. Jureczko, "Which Process Metrics Can Significantly Improve Defect Prediction Models? An Empirical Study," *Software Quality Journal*, Vol. 23, No. 3, 2015, pp. 393–422. [Online]. http://dx.doi.org/10.1007/s11219-014-9241-7

[32] M. Jureczko and J. Magott, "QualitySpy: a framework for monitoring software development processes," *Journal of Theoretical and Applied Computer Science*, Vol. 6, No. 1, 2012, pp. 35–45.

[33] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, "Comparing the effectiveness of several modeling methods for fault prediction," *Empirical Software Engineering*, Vol. 15, No. 3, 2010, pp. 277–295.

[34] L. Madeyski, *Test-driven development: An empirical evaluation of agile practice*. Springer, 2010.

[35] S. H. Kan, *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[36] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings*. IEEE, 2003, pp. 23–32.

[37] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *International Workshop on Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007*. IEEE, 2007, pp. 9–9.

[38] M. D'Ambros, A. Bacchelli, and M. Lanza, "On the impact of design flaws on software defects," in *Quality Software (QSIC), 2010 10th International Conference on*. IEEE, 2010, pp. 23–31.

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

[40] A. Bacchelli, M. D'Ambros, and M. Lanza, "Are popular classes more defect prone?" in *Fundamental Approaches to Software Engineering*. Springer, 2010, pp. 59–73.

[41] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*. ACM, 2008, p. 23.

[42] T. J. Ostrand, E. J. Weyuker, and R. M. Bell,

"Where the bugs are," in *ACM SIGSOFT Software Engineering Notes*, Vol. 29, No. 4. ACM, 2004, pp. 86–96.

[43] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, Vol. 22, No. 10, 1996, pp. 751–761.

[44] F. Brito e Abreu and W. Melo, "Evaluating the impact of object-oriented design on software quality," in *Proceedings of the 3rd International Software Metrics Symposium, 1996*. IEEE, 1996, pp. 90–99.

[45] W. L. Melo, L. Briand, and V. R. Basili, "Measuring the impact of reuse on quality and productivity in object-oriented systems," 1998.

[46] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical study of object-oriented metrics," *Journal of Object Technology*, Vol. 5, No. 8, 2006, pp. 149–173.

[47] P. Martenka and B. Walter, "Hierarchical model for evaluating software design quality," *e-Informatica Software Engineering Journal*, Vol. 4, No. 1, 2010, pp. 21–30.