Towards identifying software project clusters with regard to defect prediction

Marian Jureczko
Institute of Computer Engineering, Control and Robotics
Wrocław University of Technology
Wybrzeże Wyspiańskiego 27
50-370, Wrocław - Poland
+48 71 320 27 45
marian.jureczko@pwr.wroc.pl

Lech Madeyski
Institute of Informatics
Wrocław University of Technology
Wybrzeże Wyspiańskiego 27
50-370, Wrocław - Poland
lech.madeyski@pwr.wroc.pl
http://madeyski.e-informatyka.pl/

ABSTRACT

Background: This paper describes an analysisthat was conducted on newly collected repository with 92 versions of 38 proprietary, open-source and academic projects. A preliminary study perfomed before showed the need for a further in-depth analysis in order to identify project clusters.

Aims: The goal of this research is to perform clustering on software projects in order to identify groups of software projects with similar characteristic from the defect prediction point of view. One defect prediction model should work well for all projects that belong to such group. The existence of those groups was investigated with statistical tests and by comparing the mean value of prediction efficiency.

Method: Hierarchical and *k*-means clustering, as well as Kohonen's neural network was used to find groups of similar projects. The obtained clusters were investigated with the discriminant analysis. For each of the identified group a statistical analysis has been conducted in order to distinguish whether this group really exists. Two defect prediction models were created for each of the identified groups. The first one was based on the projects that belong to a given group, and the second one - on all the projects. Then, both models were applied to all versions of projects from the investigated group. If the predictions from the model based on projects that belong to the identified group are significantly better than the all-projects model (the mean values were compared and statistical tests were used), we conclude that the group really exists.

Results: Six different clusters were identified and the existence of two of them was statistically proven: 1) cluster proprietary B – $T=19$, $p=0.035$, $r=0.40$; 2) cluster proprietary/open – $t(17)=3.18$, $p=0.05$, $r=0.59$. The obtained effect sizes (r) represent large effects according to Cohen's benchmark, which is a substantial finding.

Conclusions: The two identified clusters were described and compared with results obtained by other researchers. The results of this work makes next step towards defining formal methods of reuse defect prediction models by identifying groups of projects within which the same defect prediction model may be used. Furthermore, a method of clustering was suggested and applied.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics – *complexity measures, product metrics, software science.*

General Terms

Measurement

Keywords

Defect Prediction, Design Metrics, Size Metrics, Clustering.

INTRODUCTION

Testing of software systems is an activity that consumes time and resources. Applying the same testing effort to all modules of a system is not the optimal approach, because the distribution of defects among individual parts of a software system is not uniform. Therefore, testers should be able to identify fault-prone classes. With such knowledge they would be able to prioritize the tests, and therefore, work more efficiently. Accorindg to Weyuker et al. [24,25] typically 20% of modules contain upwards of 80% of defects. Testers with good defect predicator may be able to spare a lot of test effort by testing only 20% of system modules and still finding up to 80% of the software defects. Defect prediction studies usually use historical data of previous versions of software to build the defect prediction models. Such approach can be applied neither in the first release of a software system, nor by companies that do not collect historical data. Therefore, it is vital to identify methods of constructing models that do not require historical data is vital.

Considerable research has been performed on the defect prediction methods; see the surveys by Purao and Vaishnavi [19], or by Wahyudin et al. and [23], but the methods of reusing of defect prediction model have not been discovered yet. There are only works where the same model has been used in similar projects (Watanabe et al. [22], Bell, Ostrand and Weyuker [2,18,24] or Nagappan et al. [16]), but without identifying the borders of similarity. According to the authors' knowledge there are only two studies where cross project validation of defect prediction models were performed [21, 26]; both are described in the next section. The goal of this research is to fulfill that gap by identifying clusters of software projects. Defect prediction in all projects that belong to one cluster should be possible to make by using only one defect prediction model. A preliminary study was already conducted [11], where existence of three clusters was investigated: proprietary projects, open-source projects and

academic projects. Only the defect prediction model created for the open-source cluster was statistically better. Therefore, only one cluster was proved to exist whereasit is extremely unlikely that the other clusters do not exist. Further studies could reveal other clusters, and it is also possible that the identified cluster may be successfully split into several smaller clusters.

The paper is organized as follows: in Section 2 related works are described. Section 3 presents the suite of OO metrics that were used, the investigated projects, definition of the study and discusses threats to validity of the study. The obtained results are shown in Section 4. Conclusions are given in Section 6 and the prospects for future research in Section 7.

RELATED WORKS

Typical approach in studies connected with defect prediction models is to build a model according to data from an old version of a project and then validate or use this model on a new version of the same project. Such approach was used [2,8,17,18,24 ,25] as well as advocated [5,23] by many researchers. Some experiments were also reported where the cross-project reusability of a defect prediction models was investigated.

Koru and Liu [12] came to interesting conclusions: "Normally, defect prediction models will change from one development environment to another according to specific defect patterns." But in their opinion, it does not mean that building generalizable defect prediction model is not possible. In fact, such models may be extremely useful and may serve as a starting point in development environments that have no historical data.

Nagappan et al. [16] extended the state of the art through analyzing whether predictors obtained from one project history are applicable to other projects. The authors investigated five proprietary software projects. The performed analyze showed that there is no single set of metrics that fits to all five projects, but the defect prediction models may be accurate when obtained form similar projects (the similarity were not precisely defined). The authors evaluated this problem by building one predictor for each project and applying it to the entities of each of the other four projects. Then the correlations between the actual and predicted rankings were compared. It turned out that the projects histories cannot serve as predictors for other projects in most cases. The study was extended in [26], where 622 cross-project predictions were performed 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 – success rate 3.4%. Subsequently, guidelines that enable assessing the chance of the success of a cross-project prediction 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.

Watanabe et al. [22] tried to apply in a C++ project a defect prediction model that has been constructed according to the data from a Java project. The reusability study in the opposite direction was conducted as well. *Sakura Editor* and *JEdit* were used as the investigated projects. Metrics from only one release were collected, so the authors stratified 10-fold cross validation model in order to count two metrics of models accuracy: precision and recall. In intra project prediction they obtained precision 0.828 and 0.733 and recall 0.897 and 0.702. In inter project prediction they obtained precision 0.872 and 0.622 and recall 0.596 and 0.402. According to obtained results, authors concluded that in the case of a similar domain and a similar size, it is possible to reuse the prediction model between languages; despite the fact the precision/recall is not very high. The authors admitted that their results were based on only two projects, so the generality is not clear and in order to increase the generalization level they were going to evaluate the reusability with other projects whose domain is text editor.

Relevant to this study are experiments conducted by Ostrand et al. [18], where two large industrial systems with separately seventeen and nine releases were investigated. A negative binomial regression model was used. The predictions were based on the source code of current release, and fault and modification history from previous release. The study was extended in [24] by analyzing the third project (it increased the number of used programming languages to ten). Applying the defect prediction model to the third project gave good results – 20% of the files that would contain the largest number of faults contained, on average, 83% of the faults. Further findings were presented in [25], where the number of the investigated projects was increased to four. According to the obtained results, the authors said: "Our prediction methodology is designed for large industrial systems with a succession of releases over years of development" but later it "was successfully adapted to a system without release". However, it must be mentioned that Weyuker et al. used another approach as the one that is presented in this paper. They had no fixed model structure, the model equation was adjusted according to data from the history of the analyzed system. Only the model building procedure was fixed.

A comprehensive study of cross company defect prediction was conducted by Turhan et al. [21]. Ten different software projects were investigated. Turhan et al. concluded that there is no single set of static code features (metrics) that may serve as defect predictor for all software projects. The defect prediction models effectiveness was measured using *probability of detection* (*pd*) and *probability of false alarm* (*pf*). Cross company defect prediction dramatically increased the *pd* as well as the *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. The project features that may influence the effectiveness of cross company predictions were not identified.

Wahyudin et al. [23] suggested a framework for defect prediction. In the context of their framework they discussed the possibility of reusing historical data in defect prediction for other projects. They concluded that: "A prediction model models the context of a particular project. As a consequence, predictors obtained from one project are usually not applicable to other projects". When the predictors are applicable or whether there exist such groups of projects within which one predicator may be applied to all projects was not discussed.

STUDY DESIGN

Metrics and Tools

There is a number of size and complexity metrics that may be used in defect prediction models. All metrics that are calculated

by the Ckjm[1] tool were used in thisstudy. The reported in [8] version of ckjm was used. This is the version that calculated 19 metrics that has been reported as good quality indicators. Those metrics were selected according to some reported experiments [3,17] and own researches [9,10]. The utilized metrics comes from several metrics suites.

The metrics suite suggested by Chidamber and Kemerer [4]:

- Weighted methods per class (**WMC**). The value of the WMC is equal to the number of methods in the class (assuming unity weights for all methods).
- Depth of Inheritance Tree (**DIT**). The DIT metric provides for each class a measure of the inheritance levels from the object hierarchy top.
- Number of Children (**NOC**). The NOC metric simply measures the number of immediate descendants of the class.
- Coupling between object classes (**CBO**). The CBO metric represents the number of classes coupled to a given class (efferent couplings and afferent couplings). These couplings can occur through method calls, field accesses, inheritance, method arguments, return types, and exceptions.
- Response for a Class (**RFC**). The RFC metric measures the number of different methods that can be executed when an object of that class receives a message. Ideally, we would want to find, for each method of the class, the methods that class will call, and repeat this for each called method, calculating what is called the transitive closure of the method call graph. This process can however be both expensive and quite inaccurate. Ckjm calculates a rough approximation to the response set by simply inspecting method calls within the class method bodies. The value of RFC is the sum of number of methods called within the class method bodies and the number of class methods. This simplification was also used in the original description of the metric.
- Lack of cohesion in methods (**LCOM**). The LCOM metric counts the sets of methods in a class that are not related through the sharing of some of the class fields. The original definition of this metric (which is the one used in Ckjm) considers all pairs of class methods. In some of these pairs both methods access at least one common field of the class, while in other pairs the two methods do not share any common field accesses. The lack of cohesion in methods is then calculated by subtracting from the number of method pairs that do not share a field access the number of method pairs that do.

One metric suggested by Henderson-Sellers [6]:

- Lack of cohesion in methods (**LCOM3**).

m - number of methods in a class;
a - number of attributes in a class;
μ(A) - number of methods that access the attribute A.

$$LCOM3 = \frac{\left(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)\right) - m}{1 - m}$$

The metrics suite suggested by Bansiy and Davis [1]:

- Number of Public Methods (**NPM**). The NPM metric simply counts all the methods in a class that are declared as public. The metric is known also as Class Interface Size (CIS)
- Data Access Metric (**DAM**). This metric is the ratio of the number of private (protected) attributes to the total number of attributes declared in the class.
- Measure of Aggregation (**MOA**). The metric measures the extent of the part-whole relationship, realized by using attributes. The metric is a count of the number of class fields whose types are user defined classes.
- Measure of Functional Abstraction (**MFA**). This metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class. The constructors and the java.lang.Object (as parent) are ignored.
- Cohesion Among Methods of Class (**CAM**). This metric computes the relatedness among methods of a class based upon the parameter list of the methods. The metric is computed using the summation of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods.

The quality oriented extension to Chidamber & Kemerer metrics suite suggested by Tang et al. [20]:

- Inheritance Coupling (**IC**). This metric provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods is functionally dependent on the new or redefined methods in the class. A class is coupled to its parent class if one of the following conditions is satisfied:
  - One of its inherited methods uses an attribute that is defined in a new/redefined method.
  - One of its inherited methods calls a redefined method.
  - One of its inherited methods is called by a redefined method and uses a parameter that is defined in the redefined method.
- Coupling Between Methods (**CBM**). The metric measures the total number of new/redefined methods to which all the inherited methods are coupled. There is a coupling when at least one of the conditions given in the IC metric is held.
- Average Method Complexity (**AMC**). This metric measures the average method size for each class. The size of a method is equal to the number of Java binary codes in the method.

Two metrics suggested by Martin [15]:

- Afferent couplings (**Ca**). The Ca metric represents the number of classes that depend upon the measured class.
- Efferent couplings (**Ce**). The Ca metric represents the number of classes that the measured class is depended upon.

One McCabe's metric [14]:

- McCabe's cyclomatic complexity (**CC**). CC is equal to the number of different paths in a method (function) plus one. The cyclomatic complexity is defined as: CC = E–N+P; where E - the number of edges of the graph, N - the number of nodes of the graph, P - the number of connected components. CC is the only method size metric. The constructed models make the class size predictions. Therefore, the metric had to be converted to a class size metric. Two metrics has been derived:

[1] http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm

- **Max(CC)** - the greatest value of CC among methods of the investigated class.
- **Avg(CC)** - the arithmetic mean of the CC value in the investigated class.

Those metrics were complemented with one more, very popular metric:

- Lines of Code (**LOC**). The LOC metric calculates the number of lines of code in the Java binary code of the class under investigation.

The information about defects occurrence was collected with a tool called BugInfo. BugInfo analyses the logs from source code repository (SVN or CVS) and according to the log content decides whether a *commit* is a bugfix. A *commit* is interpreted as a bugfix when it solves an issue reported in the bug tracking system. Each of the projects had been investigated in order to identify bugfixes commenting guidelines that were used in the source code repository. The guidelines were formalized in regular expressions. Buginfo compares the regular expressions with comments of the commits. When a comment matches the regular expression, BugInfo increments the defect count for all classes that have been modified in the commit. The BugInfo tool has had no official release yet, but we are going to implement some improvements, especially in the user interface, and then make an official release. Its current version is available at: http://kenai.com/projects/buginfo. There is no formal evaluation regarding the efficiency of this tool in mapping defects yet, but comprehensive functional tests were conducted and many of the tests are available as JUnit tests in the source code package. All collected data is available online at: http://purl.org/MarianJureczko/MetricsRepo.

Investigated projects

48 releases of 15 open source projects were investigated: *Apache Ant* (1.3 – 1.7), *Apache Camel* (1.0 – 1.6), *Ckjm* (1.8), *Apache Forrest* (0.6 – 0.8), *Apache Ivy* (1.1 – 2.0), *JEdit* (3.2.1 – 4.3), *Apache Log4j* (1.0 – 1.2), *Apache Lucene* (2.0 – 2.2), *PBeans* (1.0 and 2.0), *Apache POI* (1.5 – 3.0), *Apache Synapse* (1.0 – 1.2), *Apache Tomcat* (6.0), *Apache Velocity* (1.4 – 1.6.1), *Apache Xalan-Java* (2.4.0 – 2.7.0), *Apache Xerces* (1.1.0 – 1.4.4). A more comprehensive discussion of most of those projects was given in [8].

27 releases of 6 proprietary software projects were investigated. Five of them are custom build solutions that had been already successfully installed in the customer environment. Those five projects belong to the same domain: insurances. The 6th proprietary project is a standard tool that supports quality assurances in software development. All six projects were developed by the same company.

Moreover, 17 academic software projects were investigated. Each of them had exactly one release. Those projects were implemented by 8th or 9th semester computer science students. The students worked in groups of 3 to 6 persons during one year. A highly iterative software development process was used. A UML documentation was prepared and high level of test code coverage were obtained for each of those projects. JUnit and FitNesse were used as test tools. Some of those projects had been already investigated in [9,10].

All of the investigated projects were written in Java.

Analysis method employed

It had been assumed that character of a defect predictor strongly depends on the correlation between metrics and number of defects in a class. A correlation vector was calculated for each of the investigated releases of projects. The correlation between each of metric (the metrics are given in 3.1) and the number of defects were calculated. The vectors were than extended by adding the ratio of defects per class.

In order to uncover the project clusters, hierarchical clustering procedure and then *k*-means clustering were used. The complete linkage clustering indicated a two-group solution. Additionally Kohonen's neural network was used. The results returned by the Kohonen's neural network differ between separate runs of the network. Therefore, the network was executed several times and those releases of projects that were predominantly classified into the same neuron (cluster) were later investigated in order to distinguish whether it is a cluster from the defect prediction point of view. The obtained results were investigated with the discriminant analysis. Several different configurations of the Kohonen's network with different number of the output neurons were used, but no more than 4 clusters were obtained, even when the number of output neurons was increased up to 16.

For each of the identified cluster a defect prediction model was created. In order to create the model, all metrics were used and the stepwise linear regression was applied. Due to the stepwise regression, a typical model used five to ten metrics (not all of them). Subsequently, the models were evaluated by being applied to all releases of projects that belonged to the investigated cluster. In order to evaluate the efficiency of predicting defects in a release of project of a model, all classes that belong to the given release were sorted according to the model output. Descending predicted number of defects was used as sorting order. Next, the number of classes that must be visited in order to find 80% of defects were calculated and used as the model efficiency in predicting defects in a given release of the project. A general defect prediction model was build too. The general model used data from all the releases of all the projects as training set. In order to distinguish whether a cluster exists from the defect prediction point of view the efficiency of a model created for the cluster was compared with the efficiency of the general model. Those two models were applied only to those releases of software projects that belonged to the investigated cluster. When the efficiency of the model created for the cluster is significantly better than the efficiency of the general model one may assume that the cluster exists. In order to investigate whether the difference was significant, statistical test were used.

To render that in a more formally way, it is necessary to assume that R is a set of all releases of all projects and r is a single release of a project. C is a set of all r that were selected in a cluster. C is a subset of R ($C \subset R$). There are two defect prediction models M_R and M_C. M_R is the general model that was trained with all $r \in R$. M_C is a cluster model that was trained with all $r \in C$. $E(M,r)$ is the evaluation of efficiency of model M in predicting defects in release r. Let $c_1, c_2, \ldots, c_n$ be the classes from release r in descending order of predicted defects according to the model M_X, and $d_1, d_2, \ldots, d_n$ be the number of defects in each class. D_i is sum($d_1, \ldots, d_i$), i.e., the total defects in the first i classes. Let k be

the smallest index such that $D_k > 0.8*D_n$, then $E(M_X,r) = D_k$. $E(M_R,r)$ and $E(M_C,r)$ were calculated for all $r \in C$. In order to decide whether the cluster exists from the defect prediction point of view a hypothesis must be defined:

H_0 – There is no difference in the efficiency of defect prediction between the general model and the cluster model: $E(M_R,r)=E(M_C,r)$: $r \in C$.

H_1 – There is a difference in the efficiency of defect prediction between the general model and the cluster model: $E(M_R,r)>E(M_C,r)$: $r \in C$.

The hypotheses are evaluated by the parametric t-test for dependent samples. Following general assumptions should be checked in order to use a parametric test: level of measurement (the variables must be measured at the interval or ratio level scale), independence of observations, homogeneity of variance and the normal distribution of the sample. The homogeneity of variance is checked by Levene's test, while the assumption that the sample came from a normally distributed population is tested by the Shapiro-Wilk test [13]. When some of the assumptions are violated, the Wilcoxon matched pairs test is used.

There is an overlap between training and testing sets. In order to avoid this overlap, a separate model must be created for each of the releases from the investigated model: M_{C-r}. In such case we would get *n* different models (where *n* is the number of cluster members) and each of the models would be using different set of the releases as the training set. As a result, the definition of the cluster would be fuzzy. On the other hand, excluding one release from the training set affects the model very slightly. Therefore, we decided to use the overlaping approach.

Threats to validity

A number of limitations that may compromise to some extent the quality of the results of this study are listed below.

It is possible that there are mistakes in the defect identification. The comments in the source code version control system are not always well written and, therefore, it was sometimes very hard to decide whether a change is connected with a defect or not. In some cases the comment could be confronted with a bug tracking system, but unfortunately it was not possible in all projects.

The defects are assigned to classes according to the bugfix date. It could be probably better to assign the defect to the version, where the defect has been found, but unfortunately, the source code version control system does not contain such information.

We were not able to track operations like changing class name or moving class between packages. Therefore, after such a change, the class is interpreted as a new class. Similar difficulties were created by anonymous classes. Hence, the anonymous classes were ignored in the analysis.

The defects are identified according to the comments in the source code version control system. The guidelines of commenting bugfixes may vary among different projects. Therefore, it is possible that interpretation of the term defect is not unique among the investigated projects.

RESULTS

The results of two different approaches to clustering, using hierarchical and *k*-means clustering as well as Kohonen's neural network, are presented below.

Study 1 – two clusters

In the first study all the releases of all the projects were divided into two clusters, since the complete linkage hierarchical clustering has suggested the possibility of a "natural" partition into two sets of projects. Hence, the *k*-means two group solution is analyzed and the results are presented in Tables 1-3.

Table 1. Descriptive statistics – cluster 1st of 2

	Num. of cases	Mean	Std deviation
$E(M_R,r)$: $r \in C$	61	49.73	19.64
$E(M_C,r)$: $r \in C$	61	49.67	18.37

Table 2. Hypothesis tests – cluster 1st of 2

		$E(M_R,r)$: $r \in C$	$E(M_C,r)$: $r \in C$
Shapiro - Wilk test	W	0.987	0.991
	p	0.782	0.931
Levene's test	df	118	
	F(1,df)	0.434	
	P	0.511	
T-test	T	0.057	
	df	60	
	P	0.954	

According to Tables 1-2, the cluster *1st of 2* does not exist from the defect prediction point of view.

Table 3. Descriptive statistics – cluster 2nd of 2

	Num. of cases	Mean	Std deviation
$E(M_R,r)$: $r \in C$	31	47.18	17.80
$E(M_C,r)$: $r \in C$	31	47.41	17.29

According to Table 3, on average 47.18% of classes must be tested in order to find 80% of defects when the general model is used and 47.41% of classes when the 2nd cluster model is used. Therefore, the mean efficiency of the 2nd cluster model was worse than the mean efficiency of the general model. In consequence, there is no point in testing the hypothesis.

The conducted analysis showed that none of the two investigated clusters exists in the defect prediction point of view.

Study 2 – Kohonen's neural network

In the second approach Kohonen's neural network was used. Four clusters were identified according to the network's output. There are releases that were classified into none of those clusters.

According to Table 4, the mean efficiency of the *proprietary A* cluster model was worse than the mean efficiency of the general model. Therefore, there is no point in testing the hypothesis.

Table 4. Descriptive statistics – cluster proprietary A

	Num. of cases	Mean	Std deviation
$E(M_R,r)$: $r \in C$	11	54.16	9.54
$E(M_C,r)$: $r \in C$	11	58.18	7.73

According to Tables 5-6, there exists a cluster called *proprietary B*.

Table 5. Descriptive statistics – cluster proprietary B

	Num. of cases	Mean	Std deviation
$E(M_R,r)$: $r \in C$	14	56.35	16.59
$E(M_C,r)$: $r \in C$	14	45.05	7.81

Table 6. Hypothesis tests – cluster proprietary B

		$E(M_R,r)$: $r \in C$	$E(M_C,r)$: $r \in C$
Shapiro - Wilk test	W	0.923	0.985
	p	0.243	0.995
Levene's test	df	26	
	F(1,df)	12.778	
	p	**0,001**	
Wilcoxon matched pairs test	Z	2.103	
	T	19	
	p	**0.035**	

According to Tables 7-8, there exists a cluster called *proprietary / open.*

Table 7. Descriptive statistics – cluster proprietary / open

	Num. of cases	Mean	Std deviation
$E(M_R,r)$: $r \in C$	18	55.81	22.97
$E(M_C,r)$: $r \in C$	18	50.74	20.01

Table 8. Hypothesis tests – cluster proprietary / open

		$E(M_R,r)$: $r \in C$	$E(M_C,r)$: $r \in C$
Shapiro - Wilk test	W	0.971	0.954
	p	0.824	0.499
Levene's test	df	34	
	F(1,df)	0.155	
	p	0.696	
T-test	t	3.180	
	df	17	
	p	**0.005**	

The mean efficiency of the *open-source* cluster model was worse than the mean efficiency of the general model. Therefore, there is no point in testing the hypothesis.

Table 9. Descriptive statistics – cluster open-source

	Num. of cases	Mean	Std deviation
$E(M_R,r)$: $r \in C$	15	44.09	14.08
$E(M_C,r)$: $r \in C$	15	45.9	13.25

The existence of two clusters was proven: *proprietary B* and *open-source / proprietary*. The *proprietary B* cluster consists of custom build solutions that had been already successfully installed in the customer environment. The *open-source / proprietary* cluster consist of: Apache Forrest versions 0.7 and 0.8; Apache POI versions 2.5.1 and 3.0; Apache Xalan versions 2.4.0, 2.5.0, 2.6.0 and 2.7.0; Apache Xerces versions 1.1.0, 1.2.0, 1.3.0 and 1.4.4; jEdit versions 3.2.1, 4.1, 4.2 and 4.3; two versions of the proprietary standard tool that supports quality assurances in software development.

Discriminant Analysis

All of the identified clusters come from the Kohonen's network.

We now turn to use Fisher's linear discriminant function to derive a classification rule for assigning projects to one of the predefined groups (clusters) on the basis of the correlation vectors mentioned in Section 3.3. Means and SDs for each type of projects and overall are given in Table 10.

Table 10. Group statistics

Cluster	Mean	Std. Dev.	Cluster	Mean	Std. Dev.
2 wmc	.19	.089	4 Wmc	.13	.092
dit	-.04	.067	Dit	-.05	.048
noc	.01	.023	Noc	.00	.022
cbo	.32	.073	Cbo	.26	.098
rfc	.34	.062	Rfc	.20	.065
lcom	.13	.106	Lcom	.07	.092
ca	.18	.087	Ca	.22	.159
ce	.31	.079	Ce	.16	.147
npm	.13	.090	Npm	.10	.101
lcom3	-.05	.046	lcom3	-.04	.074
loc	.32	.070	Loc	.22	.068
dam	.02	.059	Dam	.06	.069
moa	.04	.054	Moa	.04	.025
mfa	-.07	.054	Mfa	-.04	.073
cam	-.15	.053	Cam	-.10	.063
ic	-.02	.052	Ic	.02	.127
cbm	-.01	.035	cbm	.02	.085
amc	.18	.056	amc	.14	.132
max_cc	.18	.048	max_cc	.15	.089
avg_cc	.13	.050	avg_cc	.09	.095
bugs/classes	.24	.202	bugs/classes	.18	.157

3	wmc	.35	.152	5	wmc	.45	.221
	dit	-.01	.082		dit	.01	.067
	noc	.01	.065		noc	.06	.096
	cbo	.30	.176		cbo	.29	.177
	rfc	.46	.193		rfc	.50	.218
	lcom	.33	.184		lcom	.40	.262
	ca	.13	.227		ca	.16	.142
	ce	.37	.181		ce	.41	.164
	npm	.32	.215		npm	.40	.211
	lcom3	-.08	.078		lcom3	-.06	.054
	loc	.34	.125		loc	.44	.208
	dam	.12	.072		dam	.11	.073
	moa	.30	.156		moa	.29	.134
	mfa	-.02	.075		mfa	-.02	.070
	cam	-.20	.096		cam	-.25	.097
	ic	.07	.114		ic	.08	.102
	cbm	.12	.134		cbm	.09	.098
	amc	.09	.104		amc	.11	.091
	max_cc	.18	.161		max_cc	.20	.144
	avg_cc	.10	.125		avg_cc	.11	.090
	bugs/classes	.73	.669		bugs/classes	.71	.697

Box's test for equality of covariance cannot be performed due to fewer than two nonsingular group covariance matrices. However, even if Box's test suggests a departure for the equality hypothesis, the linear discriminant may still be preferable over a quadratic function. Here we will assume normality for our data relying on the robustness of Fisher's approach to deal with any minor departure from the assumption [7].

The eigenvalues (here 4.66, 3.13 and .76 presented in Table 11) represent the ratios of the between-group sums of squares to the within-group sum of squares of the discriminant scores. The canonical (Pearson) correlations are correlations between the discriminant function scores and group membership coded as 2 (*cluster proprietary B*), 3 (*cluster proprietary / open*), 4 (*cluster proprietary A*) and 5 (*cluster open source*). The canonical correlation values (0.907, 0.871 and 0.656) are presented in Table 11. First 3 canonical discriminant functions were used in the analysis.

Table 11. Eigenvalues

Function	Eigenvalue	% of Variance	Cumulative %	Canonical Correlation
1	4.663[a]	54.5	54.5	0.907
2	3.132[a]	36.6	91.1	0.871
3	0.757[a]	8.9	100.0	0.656

As a result, 82.2%, 75.9% and 43% of the variance in the discriminant function scores can be explained by group differences. The Wilk's Lambda, presented in Table 12, provides a test for assessing the null hypotheses that in the population the vectors of means of the measurements are the same in groups.

Table 12. Wilks' Lambda

Test of Function(s)	Wilks' Lambda	Chi-square	df	Sig.
1 through 3	0.024	157.939	63	0.000
2 through 3	0.138	84.249	40	0.000
3	0.569	23.955	19	0.198

The lambda coefficient is defined as the proportion of the total variance in the discriminant scores not explained by differences among groups, here 0.02%, 13.8% and 56.9% respectively. The formal test confirms that the sets of measurements (correlations between software metrics and faulty classes) differ significantly between the clusters with exception of the last one (Chi-square (63)=157.939, p=0.000; Chi-square(40)=84.249, p=0.000; Chi-square(19)=23.955, p=0.198). An important question about a discriminant function is: how well does it perform? According to the obtained results presented in Table 13, 96.4% of cases can be correctly classified as type 2, 3, 4 or 5. However, estimating misclassification rates in this way is known to be too optimistic and different alternatives for estimating misclassification rates in discriminant analysis have been proposed. *Leaving one out method* is one of these alternatives in which the discriminant function is first derived from only *n* - 1 sample members, and then used to classify the observation left out. The aforementioned procedure is repeated *n* times, each time omitting a different observation. As a result, the classification rate drops to 71.4%.

Table 13. Classification Results

Classification Results[b,c]

			Predicted Group Membership				
		clusterJur4	2	3	4	5	Total
Original	Count	2	15	0	0	0	15
		3	1	17	0	0	18
		4	0	0	8	0	8
		5	0	1	0	14	15
		Ungrouped cases	5	14	2	15	36
	%	2	100.0	.0	.0	.0	100.0
		3	5.6	94.4	.0	.0	100.0
		4	.0	.0	100.0	.0	100.0
		5	.0	6.7	.0	93.3	100.0
		Ungrouped cases	13.9	38.9	5.6	41.7	100.0
Cross-validated[a]	Count	2	13	0	2	0	15
		3	1	12	2	3	18
		4	3	1	4	0	8
		5	1	3	0	11	15
	%	2	86.7	.0	13.3	.0	100.0
		3	5.6	66.7	11.1	16.7	100.0
		4	37.5	12.5	50.0	.0	100.0
		5	6.7	20.0	.0	73.3	100.0

a. Cross validation is done only for those cases in the analysis. In cross validation, each case is classified by the functions derived from all cases other than that case.

b. 96.4% of original grouped cases correctly classified.

c. 71.4% of cross-validated grouped cases correctly classified.

PRACTICAL IMPLICATION

The conducted analysis revealed two clusters. In order to increase the value of those findings, the characteristic of those clusters must be given and the obtained results should be compared with the other studies.

Characteristic of revealed clusters

Cluster proprietary B. This cluster consists of slightly more than half of the proprietary custom solutions. All custom build

solutions (not only those that belongs to the *proprietary B* cluster) were developed in heavy weight, plan-driven development process, all of them had already been successfully installed in the customer environment (the investigated releases were developed after the installation, but the releases consisted not only of bugfixes - there were included many new features as well) and all of them came from the same domain – insurances. The interviews with people involved in the development process of the proprietary projects were conducted in order to find the factor that might settle whether a project release belongs to the *proprietary B* cluster or not. The interviews revealed that there was a difference in the testing process on the level of functional system tests. Almost all the releases from cluster *proprietary B* were tested manually, when almost all the releases that were not selected to the cluster *proprietary B* were tested automatically. The testing factor explained cluster membership of all except three releases. There were three releases that were tested manually, but were not selected to the cluster *proprietary B*. The absence of those three releases was easy to explain. Those three releases had much smaller defect rate, which was a result of a shorter development period.

Cluster proprietary / open. The cluster consist of: Apache Forrest versions 0.7 and 0.8; Apache POI versions 2.5.1 and 3.0; Apache Xalan versions 2.4.0, 2.5.0, 2.6.0 and 2.7.0; Apache Xerces versions 1.1.0, 1.2.0, 1.3.0 and 1.4.4; jEdit versions 3.2.1, 4.1, 4.2 and 4.3; two versions of the proprietary standard tool that supports quality assurances in software development. This cluster consists of several different projects that were developed by different companies in different software development processes. The domain of those projects is not uniform, but all of them are more or less connected with text processing. All members of this cluster, except JEdit, use Jira or Bugzilla (or both) as the bug tracking system, all (again except JEdit) are developed by medium size international team – the greater team consists of 25 persons, but in most cases it was exactly 11 persons. High level of automatization in the testing process (the data about testing process were not available for all releases) was applied in most cases, and in all of them SVN repositories were used as the source code version control system.

Common features of the identified clusters are summarized in Table 14.

Table 14. Common features of the identified clusters

Cluster	Common features
Cluster proprietary B	custom build solutions; heavy weight, plan-driven development process; already installed in the customer environment; insurance domain; manual tests; similar development period; use databse; proprietary – the same company
Cluster proprietary / open	text processing domain; SVN and Jira or Bugzilla used; medium size international team; automatization in the testing process; do not use database

Comparison with other studies

Zimmermann et al. [25] provided a decision tree that helps to evaluate the precision of cross projects defect predictions. If our results are compatible with the Zimmermann's findings then many cluster members should share the features that, according to the decision tree, increase the precision and few of them should share features that - according to the decision tree - decrease the precision.

According to the decision tree, projects or releases with more or the same *Number of observations* (in our case that is the number of classes) should better predict defects in projects with fewer *Number of observations*. This finding does not fit in the concept of clusters. All the members of a well identified cluster should be transitive and this finding is strongly connected with lack of transitivity – in [25] were identified such projects that project *A* was a good defect predictor for project *B*, but project *B* was a bad defect predictor for the project *A*. The *Number of observations* finding can be only partly verified. Projects with equal size should be good defect predictors for each other. The size of releases from *cluster proprietary B* varies from 2286 to 4057, but there are also two outliers: 1694 and 4622. The relative difference is not big in the cluster. *Number of cases* in releases selected to cluster *proprietary / open* vary from 162 to 909 with two outliers: 29 and 32.

The second factor that, influences defect prediction precision is according to [25], *Uses database*. When both projects do not use the database, the prediction precision should be increased. All members of the *cluster proprietary B* use database. None of the members of the *cluster proprietary / open* uses database.

The third factor according to [25] is *Company*. Zimmermann et al. listed peculiar companies that have positive or negative influence on prediction precision. Only two factors are relevant for us: cross prediction within Apache products increases the precision and cross company prediction decreases the precision. All members of the *proprietary B cluster* come from the same company – Capgemini-sd&m. 12 members of the *cluster proprietary / open* come from Apache, 4 from JEdit community and 2 from Capgemini-sd&m.

The adequacy of clustering

The discriminant analysis showed that the clusters explain most of the variety. Discriminant analysis of clusters obtained from Kohonen's network is given in 4.4. Analysis of other clusters was omitted because of the lack of space, but was also conducted and the obtained results were, according to the discriminant analysis, even more adequate. The clustering was based on correlation vectors. One may argue if the correlation vectors are the optimal base for clustering because the effectiveness of defect prediction models built for most of the clusters was not significantly better than the effectiveness of the general model.

CONCLUSIONS

Metrics from 92 releases of 38 proprietary, open-source and academic projects were collected, stored in a repository (http://purl.org/MarianJureczko/MetricsRepo) and analyzed. The conducted analysis reveals that there exist clusters from the defect prediction point of view and two of those clusters were successfully identified. The existence of those two clusters was proven with statistical tests. The features of cluster members were described and compared with findings of other studies. The comparison showed that there are no major inconsistencies with the other studies and our findings partly overlap the results obtained by other researchers [25]. Reproducing the study in an

industrial environment is difficult because in order to construct the correlation vectors the information about defects (that one is going to predict) is necessary. Therefore, we strongly recommend using the factors presented in section 5.1 as cluster indicators instead of the correlation vectors.

The obtained results are not astonishing. The existence of only two clusters was proven. According to other studies the cross project defect prediction is a complicated issue. Zimmermann et al. [25] reported success rate 3.4% in cross project predictions. Turhan et al. [21] obtained *probability of false alarm* greater than 50% in most of the cross project predictions. Therefore the presented results may be considered as interesting ones, but hey definitely point to the need of further research.

FUTURE RESEARCH

The identified clusters are far away from covering all software projects. Further research is necessary to identify more clusters. The clusters that were identified are very wide and therefore it is possible that those clusters may be successfully divided into smaller ones. In both cases, it is necessary to collect and analyze more data about software projects in order to reach those goals.

There may be conducted a cross validation for the study. One may build one defect prediction model per each release and then use the models to predict defects for other releases. If the clusters are correctly identified the within cluster prediction will be better than the cross cluster predictions. That approach would be similar to the one that was used in [22,25]. With such approach, it is possible to identify outliers that were classified to a cluster, but do not fit to the cluster very well.

ACKNOWLEDGMENTS

The authors are very grateful to the Capgemini Polska Company that allowed analyzing five of their proprietary projects. Thus, the research has been better validated – the authors could use not only open source, but also industrial projects.

REFERENCES

[1] Bansiya, J. and Davis, C. G. 2002. A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Trans. on Software Engineering, 28, 1, (January 2002). 4-17. DOI= http://doi.ieeecomputersociety.org/10.1109/32.979986

[2] Bell, R., Ostrand T., Weyuker E. 2006. Looking for Bugs in all the right places. In Proceeding of the 2006 International Symposium on Software Testing and Analysis (Portland, USA, July 17-20, 2006). ISSTA 2006. ACM Press New Your, NY, 61-72. DOI= http://doi.acm.org/10.1145/1146238.1146246

[3] Catal, C., Diri, B., Ozumut, B. 2007. An Artificial Immune System Approach for Fault Prediction in Object-Oriented Software. In Proceedings of the 2nd International Conference on Dependability of Computer Systems (Szklarska Poręba, Poland, 14-16 June, 2007). DepCoS-RELCOMEX 2007. IEEE. 238-245. DOI= http://doi.ieeecomputersociety.org/10.1109/DEPCOS-RELCOMEX.2007.8

[4] Chidamber, S. R., and Kemerer, C. F. A metrics suite for object oriented design. IEEE Transaction on Software Engineering, 20, 6, (June 1994). 476–493. DOI= http://doi.ieeecomputersociety.org/10.1109/32.295895

[5] Fenton, E., Neil, M. 1999. A Critique of Software Defect Prediction Models. IEEE Transaction on Software Engineering, 25, 5, (September 1999). 675-689. DOI= http://dx.doi.org/10.1109/32.815326

[6] Henderson-Sellers, B., Object-Oriented Metrics, measures of Complexity. Prentice Hall, 1996.

[7] Hand, D. J. 1998. Discriminant Analysis, Linear. In Encyclopedia of Biostatistics Volume 2 (P. Armitage and T. Colton, Eds.). Chichester: Wiley.

[8] Jureczko, M., Spinellis D. 2010. Using Object-Oriented Design Metrics to Predict Software Defects. In Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej. 69-81.

[9] Jureczko, M. 2007. Use of software metrics for finding weak points of object oriented projects. Proceeding of Metody i narzędzia wytwarzania oprogramowania (Szklarska Poręba, Poland, 14-16 May, 2007). 133-144.

[10] Jureczko, M., 2008. Ocena jakości obiektowo zorientowanego projektu programistycznego na podstawie metryk oprogramowania. In Inżynieria oprogramowania – metody wytwarzania i wybrane zastosowania. PWN. 364-377.

[11] Jureczko, M., Madeyski, L. 2010. Predykcja defektów na podstawie metryk oprogramowania – identyfikacja klas projektów. Submitted on KKIO 2010.

[12] Koru, A.G., Liu, H. 2005. Building Defect Prediction Models in Practice. IEEE Software 22, 6, (December 2005). 23-29. DOI= http://dx.doi.org/10.1109/MS.2005.149

[13] Madeyski, L., Test-Driven Development: An Empirical Evaluation of Agile Practice. Springer, 2010. DOI= http://dx.doi.org/10.1007/978-3-642-04288-1

[14] McCabe, T. J. 1976. A complexity measure. IEEE Trans. on Software Engineering, 2, 4, (1976). 308-320.

[15] Martin, R. 1994. OO Design Quality Metrics - An Analysis of Dependencies. Proceeding of Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA'94.

[16] Nagappan N., Ball T., Zeller A. 2006. Mining Metrics to Predict Component Failures. In Proceedings of the 28th International Conference on Software Engineering. (Shanghai, China, May 20-28, 2006). ICSE'06. ACM Press New Your, NY, 452-461. DOI= http://doi.acm.org/10.1145/1134285.1134349

[17] Olague, H., Etzkorn, L., Gholston S., Quattlebaum S. 2007. Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes. IEEE Transaction on Software Engineering 33, 6, (June 2007). 402-419. DOI= http://doi.ieeecomputersociety.org/10.1109/TSE.2007.1015

[18] Ostrand, T., Weyuker, E., Bell, R. 2005. Predicting the Location and Number of Faults in Large Software Systems.

IEEE Trans. on Software Engineering 31, 4, (April 2005). 340-355. DOI= http://dx.doi.org/10.1109/TSE.2005.49

[19] Purao, S. and Vaishnavi V. K. 2003. Product metrics for object-oriented systems. ACM Computing Surveys 35, 2 (June 2003).191-221. DOI=http://doi.acm.org/10.1145/857076.857090

[20] Tang, M-H., Kao, M-H., and Chen, M-H. 1999. An Empirical Study on Object-Oriented Metrics. In Proceedings of the Sixth International Software Metrics Symposium (Boca Raton, USA, 4-6 November, 1999). 242-249. DOI= http://dx.doi.org/10.1109/METRIC.1999.809718

[21] Turhan, B., Menzies, T., Bener, A., Distefano, J. 2009. On the Relative Value of Cross-Company and Within-Company Data for Defect Prediction. Empirical Software Engineering 14, 5, (October 2009). 540-578. DOI=http://dx.doi.org/10.1007/s10664-008-9103-7

[22] Watanabe, S., Kaiya, H., Kaijiri K. 2008. Adapting a Fault Prediction Model to Allow Inter Language Reuse. In Proceedings of the 4th International Workshop on Predictive Models in Software Engineering (Leipzig, Germany, May 12-13, 2008). PROMISE'08.

[23] Wahyudin, D., Ramler, R. and Biffl S. 2008. A framework for Defect Prediction in Specific Software Project Contexts. In Proceedings of the 3rd IFIP Central and East European Conference on Software Engineering Techniques (Brno, Czech Republic, October 13-15, 2008). CEE-SET 2008.

[24] Weyuker, E., Ostrand T., Bell R. 2008. Adapting a Fault Prediction Model to Allow Widespread Usage. In Proceedings of the the International Workshop on Predictive Models in Software Engineering (Leipzig, Germany, May 12-13, 2008). PROMISE'08.

[25] Weyuker, E., Ostrand T., Bell R. 2008. Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models. Empirical Software Engineering, 13, 5 (October 2008). 539-559. DOI= http://dx.doi.org/10.1007/s10664-008-9082-8

[26] Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B. 2009. Cross-project Defect Prediction. In Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (Amsterdam, The Netherlands, August 24-28 2009). ESEC/FSE 2009. 91-100.