The Stability of Threshold Values for Software Metrics in Software Defect Prediction

Goran Mauša(✉) and Tihana Galinac Grbac

Faculty of Engineering, University of Rijeka, Vukovarska 58, 51000 Rijeka, Croatia
goran.mausa@riteh.hr, tihana.galinac@rireh.hr

Abstract. Software metrics measure the complexity and quality in many empirical case studies. Recent studies have shown that threshold values can be detected for some metrics and used to predict defect-prone system modules. The goal of this paper is to empirically validate the stability of threshold values. Our aim is to analyze a wider set of software metrics than it has been previously reported and to perform the analysis in the context of different levels of data imbalance. We replicate the case study of deriving thresholds for software metrics using a statistical model based on logistic regression. Furthermore, we analyze threshold stability in the context of varying level of data imbalance. The methodology is validated using a great number of subsequent releases of open source projects. We revealed that threshold values of some metrics could be used to effectively predict defect-prone modules. Moreover, threshold values of some metrics may be influenced by the level of data imbalance. The results of this case study give a valuable insight into the importance of software metrics and the presented methodology may also be used by software quality assurance practitioners.

Keywords: Software metrics · Threshold · Data imbalance · Software defect prediction

1 Introduction

There are many different software metrics and each describes the program code from a different perspective [1,2]. Software metrics that describe the object oriented code can be used to quantify its quality [3]. One of the most direct attributes of software quality is the number of defects that need to be handled after the implementation phase. Software defect prediction (SDP) is the research area that investigates the possibility to use these metrics to classify the software modules that contain defects. Contradictory results have been reported across studies that evaluated the classification accuracy of various software metrics [4]. There are indications that the choice of the prediction model has lower impact on accuracy than the choice of metrics [5]. However, certain software metrics proved to be efficient for defect prediction [6]. Moreover, some metrics exhibit threshold effect, i.e. a relationship can be identified between a metric's threshold value and the occurrence of defects [7]. Threshold is the level of a metric, above which we classify the module to be faulty, i.e. to contain defects, and non-faulty otherwise.

Software defects are generally not distributed according to any probability distribution that could be described with a particular mathematical model [8]. That is why SDP community turned to machine learning methods. However, there is not any prediction model that is the best in all the application contexts [9]. Furthermore, most machine learning methods exhibit performance deterioration in high levels of data imbalance [10]. Data imbalance is the phenomenon in which one class of data greatly outnumber the other class or classes of data. The class that represents faulty software modules is outnumbered in SDP, making data imbalance its inherent feature [11]. Recent studies experimented with the use of genetic programming algorithms for classification, i.e. building the prediction model in SDP, and achieved some promising results [12]. The multi-objective framework of genetic programming improves the generalization ability by forming ensembles of diverse classifiers [13]. These algorithms combine all the metrics in a decision tree structure and train the prediction models using multiple fitness functions sensitive to data imbalance. In such configuration, every software metric is one additional dimension in the search space. Having a large search space with too many dimensions reduces the probability of finding the optimal solutions in any heuristic optimization algorithm [14]. Furthermore, these algorithms may be further improved by incorporating certain domain knowledge [15]. This may be done by using a predefined initial populations or by fine tuning its configuration.

This paper aims to empirically validate the stability of threshold values of software metrics for SDP. We replicate the threshold derivation model from the studies performed by [7,16]. The stability of threshold values is examined in terms of rate of significance, spread of threshold values and the difference of central tendency between different datasets. The results of this case study research provides us with insight into the importance of individual software metric for SDP. This valuable domain knowledge will be used to improve the genetic algorithm configuration that achieved promising results in spite of data imbalance [12]. Software metrics not significant for defects prediction will be excluded and the significant ones will be given higher priority in the decision making process. Moreover, the threshold values of metrics that prove to be effective in prediction will be regarded as reference points in a multivariate model configuration. That is why it is important to perform a large scale case study and include all the possible software metrics, whereas previous research focused mainly on a small number of object-oriented metrics [7,17]. This paper performs this study on 49 different metrics that are calculated according to the systematically defined collection procedure for SDP research [18].

The remainder of this paper is structured as follows: Sect. 2 presents an overview of related work; Sect. 3 describes the details of the case study we have conducted; Sect. 4 shows and discusses the results that we have obtained; Sect. 5 gives a conclusion and explains our future work intentions.

2 Related Work

Many studies searched for software metrics with strong association to software defect prediction. Majority of these studies concentrated on a small number of object oriented metrics like: coupling between objects (‘CBO’), response for class (‘RFC’), weighted method per class (‘WMC’), depth of inheritance tree (‘DIT’) and number of children (‘NOC’) [7]. Their results indicated that metrics like ‘CBO’, ‘WMC’ and ‘RFC’ were often significant for SDP and other metrics like ‘DIT’ and ‘NOC’ were rarely significant [7,16]. A recent replicated study involved other metrics like: lack of cohesion between objects (‘LCOM’), maximum or average cyclomatic complexity (‘MAXCC’, ‘AVCC’) or lines of code (‘LOC’) and demonstrated that they may also be significant [16].

On the other hand, fewer studies experimented with threshold derivation [16]. That does not mean that practitioners do not use it. The senior software developers usually determine the threshold values based on their experience [19]. Their decision making process cannot be replicated nor reused and it is highly biased. Hence, researchers proposed several methods to perform it systematically. The most popular are the Bender method which is based on logistic prediction model and the method which is based on receiver operating curve (ROC) [16].

In most classification applications that suffer from data imbalance, the minority class is usually the one that is more important to find and this posses a problem to most classification algorithms [20]. The fact that defects in large and complex software systems are distributed according to the Pareto principle [11,21,22] makes data imbalance an inherent feature of SDP. The rate of faulty software modules is always lower than the rate of modules that are non-faulty and we refer to it as the level of imbalance in the rest of this paper. Different methods were proposed to deal with data imbalance. The ones that take into account the misclassification costs of unequally distributed classes were generally the most successful [12,23]. In this paper we analyze whether data imbalance has an impact on threshold derivation method as well and use the proper evaluation metrics to take that into account.

3 Case Study

The goal of this paper is to empirically validate the stability of threshold values in different contexts. We want to see whether there are metrics significant for defect prediction with stable threshold values so that the practitioners could use them in software quality assurance. The threshold values are calculated by using the logistic regression model and Bender method, and evaluated by using the decision tree binary classification. The proposed methodology examines individually the strength of each metric for defect prediction. The stability of threshold values is examined in datasets with varying levels of imbalance by using 10-fold cross-validation. We used Matlab R2014a to perform the case study calculations.

3.1 Data

We use 14 datasets of two open source projects from the Eclipse community: JDT and PDE. Each dataset represents one release and there are 7 subsequent releases of each project in this case study. We used the BuCo Analyzer tool [24] to collect the datasets. BuCo is a tool that implements systematically defined data collection procedure [18]. The tool performs linking between bugs and commits using a technique that is based on regular expression [25]. The effectiveness of the linking is expressed by linking rate, i.e. the ratio of bugs that are linked to at least one commit. We present the linking rate, the number of reported bugs, the number of Java files, the ratio between non-faulty and faulty files and the number of software metrics for each dataset in Table 1.

Table 1. Datasets used in this study

Eclipse JDT project
tool plug-ins that support the development of any Java application

Release	Linking Rate	Bugs	Files	Non-Faulty : Faulty	Metrics
2.0	48.4%	4276	2397	54.1% : 45.9%	50
2.1	64.4%	1875	2743	68.1% : 31.9%	50
3.0	70.9%	3385	3420	61.4% : 38.6%	50
3.1	80.6%	2653	3883	67.3% : 32.7%	50
3.2	84.9%	1879	2233	63.5% : 36.5%	50
3.3	88.7%	1341	4821	76.2% : 23.8%	50
3.4	90.0%	989	4932	81.5% : 18.5%	50

Eclipse PDE project
tools to create, develop, test, debug, build and deploy Eclipse plug-ins

Release	Linking Rate	Bugs	Files	Non-Faulty : Faulty	Metrics
2.0	22.3%	561	576	80.7% : 19.3%	50
2.1	27.4%	427	761	83.7% : 16.3%	50
3.0	33.6%	1041	881	68.8% : 31.2%	50
3.1	51.6%	769	1108	67.8% : 32.2%	50
3.2	69.2%	546	1351	53.8% : 46.2%	50
3.3	85.3%	727	1713	56.3% : 43.7%	50
3.4	80.9%	963	2144	71.5% : 28.5%	50

3.2 Methodology

The methodology of this case study is presented in Fig. 1. In the first phase we sample the datasets into 10 equally distributed folds, according to the stratified sampling strategy of 10-fold cross-validation. We use the 10-fold cross-validation to validate the stability of the results that are going to be obtained in the following releases. The training dataset contains 9 folds and the testing dataset contains the remaining fold. This process is repeated 10 times, each time with

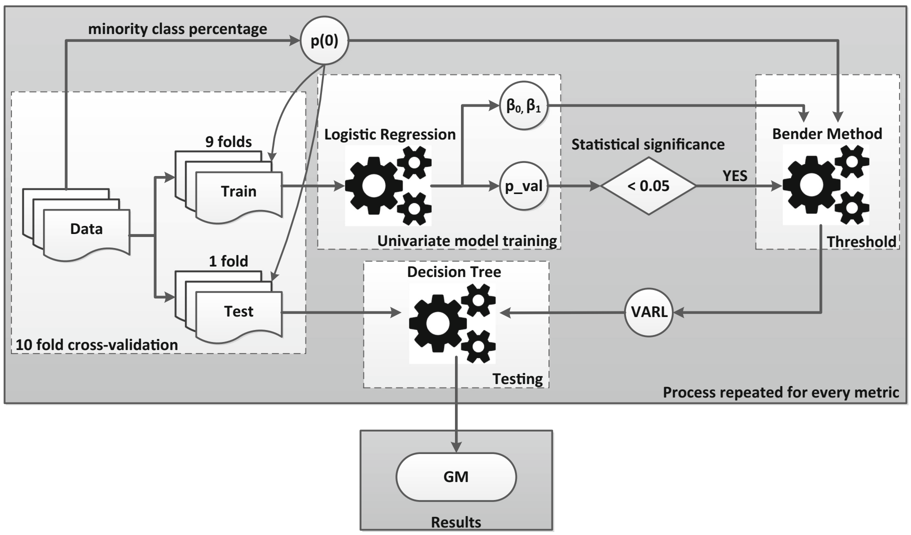

Fig. 1. Methodology of this case study

different fold as the testing dataset but with equal ratio between the minorty and the majority class.

The following three phases belong to the threshold derivation model. The univariate logistic regression model is built for each dataset separately in the second phase. The logistic regression model is presented with the following equation [26]:

$$P\left(X\right)=\frac{e^{\beta_0+\beta_1\cdot X}}{1+e^{\beta_0+\beta_1\cdot X}} \tag{1}$$

where β_0 is the free coefficient, β_1 is the regression slope coefficients for predictor X, X represents a particular metric and $P(X)$ is the probability that a module is faulty. The coefficients β_0 and β_1 define the curvature and the non-linearity of the logistic regression output curve.

Base probability $\mathbf{p(0)}$ is the cutoff value that is required to transform probability $P(X)$ into binary variable. It is often set to 0.5 [27], but it can be tuned to account for data imbalance present in the dataset [3]. Using the percentage of the majority class as the base probability accounts for the higher probability that a randomly selected module is non-faulty [3]. This is also applicable to the Bender method in the following phase so the majority class percentage is forwarded there. An intermediate step in the methodology is also the analysis of statistical significance. The null-hypothesis states there is no relationship between the logistic regression model of a particular metric and the defect-proneness of the examined software modules. This step forwards the metrics for which the null-hypothesis is rejected to the following phase.

We calculate the value of an acceptable risk level, i.e. the threshold values for each metric THR_j, where j represents the metric, in the third phase. For the *significant* metrics, thresholds are derived using the following equation given by the Bender method [28]:

$$THR_j = \frac{1}{\beta_{1,j}}(ln(\frac{p_0}{1-p_0}) - \beta_{0,j}), \tag{2}$$

where β_0 and β_1 are coefficients obtained from the logistic regression model and $p(0)$ is the base probability. As explained earlier, we used the percentage of majority class as the base probability ($p_0 = p(0)$) [16].

We evaluate the effectiveness of every threshold in the fourth phase. A simple decision tree is built using the obtained threshold values to classify the software modules from the testing dataset according to the following equation:

$$Y_i = \begin{cases} faulty & if\ X_{i,j} > THR_j \\ non-faulty & if\ X_{i,j} > THR_j \end{cases}, \tag{3}$$

where i and j are the software module and the software metric indexes, respectively. In our study, the granularity level of software modules is the file level. The software metrics in our study describe the size and complexity of source code (like lines of code, cyclomatic complexity), the usage of object oriented principles (inheritance, encapsulation, abstraction and polymorphism), the design of source code (like coupling and cohesion), the programming style (like number of comments, blank lines of code) and more. The full list of metrics and their descriptions that we have used can be found in [12]. Then we compare the output of prediction with the actual values and evaluate its performance. The following subsection describes this in more details.

3.3 Evaluation

General importance of each metric for the defect prediction is evaluated upon building the model based on logistic regression. We test the statistical significance in a univariate model by making the opposite null-hypothesis that there is no relationship between the calculated coefficient and the defect-proneness of the modules. If the p-value is lower than 0.05, we reject the null-hypothesis and conclude that the observed metric is significant for prediction.

A more precise evaluation of the metrics that are statistically significant for prediction is done after building the decision trees. There are four possible outcomes when performing binary classification and they are present with the confusion matrix in Table 2. The faulty software modules constitute the positive (1) class, while non-faulty ones constitute the negative (0) class.

There are various evaluation metrics that can be computed using the true positive (TP), true negative (TN), false positive (FP) and false negative (FN) predictions. In this paper, we use the *geometric mean accuracy* (GM) to evaluate the performance of the decision tree classifier that is based on calculated level

Table 2. Confusion matrix for binary classification

Actual Value:	Predicted Value: Faulty (1)	Predicted Value: Non-Faulty(0)
Faulty(1)	**TP**	**FN**
Non-Faulty(0)	**FP**	**TN**

of threshold for each metric. GM is calculated as the geometric mean using the following equation:

$$GM = \sqrt{TPR \cdot TNR}, \tag{4}$$

where TPR (true positive rate) and TNR (true negative rate) are calculated as:

$$\begin{aligned} TPR &= TP/\left(TP + FN\right), \\ TNR &= TN/\left(TN + FP\right). \end{aligned} \tag{5}$$

Its values are within a range of [0,1], higher results indicating better performance. The main advantage of this evaluation metric is its sensitivity to class imbalance [29]. We adopt the interpretation that metrics which achieve $GM > 0.6$ are considered effective for prediction [7,16].

4 Results

The first step in our case study was to build an univariate logistic regression model for each metric. We used a function built in the Matlab system for that. The statistical significance of computed coefficients is one of the outputs of that function. Metrics that had *p-value* > 0.05 were not considered relevant for defect prediction and were reject from following analysis. Table 3 presents how many times the null-hypothesis was rejected in each of the analyzed releases. The maximum number of times a metric could be significant is 10 because we used 10-fold cross-validation. Column named *sum* presents the average rate of significance in the project. The table is vertically divided in two parts, left part representing JDT releases and right part representing PDE releases. The metrics are placed in rows and we categorized several types of metrics in the table based on their summed values:

– Metrics significant in every release of both projects are marked **bold**,
– Metrics significant in majority of releases of both projects are not marked,
– Metrics rarely significant in both projects are marked in dark gray color,
– Metrics significant in every release of one project and rarely significant in the other are marked in light gray color.

Threshold values are calculated and evaluated in terms of GM each time a metric is considered significant. We have analyzed the distribution of GM values for all the datasets. Due to a great number of datasets and space limit, we

Table 3. Rate of rejecting the null-hypothesis in all datasets

Project:	JDT								PDE							
Release:	2.0.	2.1.	3.0.	3.1.	3.2.	3.3.	3.4.	sum	2.0.	2.1.	3.0.	3.1.	3.2.	3.3.	3.4.	sum
'LOC'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'SLOC_P'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'SLOC_L'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'MVG'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'BLOC'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'C_SLOC'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'CLOC'	10	10	10	10	10	10	10	100%	0	0	0.1	3	0	5	10	27%
'CWORD'	10	10	10	10	10	10	10	100%	0	0	0.1	0	5	10	10	37%
'HCLOC'	0	3	0	0	0	0	0	4%	8	0	0	0	0	0	10	26%
'HCWORD'	10	9	0	0	0	0	9	40%	7	0	10	0	0	0	10	39%
'No_Methods'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'LCOM'	0	0	0	0	0	0	0	0%	10	10	8	10	10	10	10	97%
'AVCC'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'NOS'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'HBUG'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'HEFF'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'UWCS'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'INST'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'PACK'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'RFC'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'CBO'	10	10	10	10	10	10	10	100%	0	0	0	0	0	0	0	0%
'MI'	10	10	10	10	10	10	10	100%	10	10	10	10	6	0	0	66%
'CCML'	10	10	10	10	10	10	10	100%	3	0.1	10	10	10	10	10	77%
'NLOC'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'F_IN'	10	10	10	10	10	10	10	100%	0	0	0	0	0	0	0	0%
'DIT'	0	0	0.1	0	5	0	10	23%	0	0	0.1	0	0	0	0	1%
'MINC'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'S_R'	0	0	0	0.1	0	6	10	24%	0	0	0	0	0	0	0	0%
'R_R'	10	10	10	10	10	10	10	100%	10	0	6	10	10	10	10	80%
'COH'	0	0	0	0	10	0	10	29%	10	6	0	0	0	0	5	30%
'LMC'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'LCOM2'	10	10	10	10	9	10	10	99%	10	10	8	10	10	10	10	97%
'MAXCC'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'HVOL'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'HIER'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'NQU'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'FOUT'	0	0	2	0	0	5	10	24%	0	0	2	0.1	0	0	0	4%
'SIX'	0	0	0	0.1	8	0	10	27%	0	0	0	0	0	0	0	0%
'EXT'	10	10	10	10	10	10	10	100%	0	0	2	10	10	10	10	60%
'NSUP'	10	10	10	10	10	10	10	100%	10	0	8	10	10	10	8	80%
'TCC'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'NSUB'	0	0	3	0.1	2	9	10	36%	0	0	0	0	0	0	0	0%
'MPC'	10	10	10	10	10	10	10	100%	0	0	2	10	10	10	10	60%
'NCO'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'INTR'	10	10	10	10	10	10	10	100%	10	2	10	10	10	10	10	89%
'CCOM'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'HLTH'	10	10	10	10	10	10	10	100%	10	10	10	10	10	10	10	100%
'MOD'	10	10	10	10	10	10	8	97%	3	0	0	0	0	0	0	4%

present the distribution of results only for all releases together. The box and whisker plots is given in Figs. 2 and 3. The central mark presents the median, the box is interquartile range, the whiskers extend to the non-outlier range, and outliers are plotted individually. Due to a great number of metrics, we omitted the ones that did not have 100% rate of significance in the previous analysis. The majority of presented metrics is considered effective, according to the ($GM > 0.6$) interpretation. Figure 2 presents the results obtained from the releases of JDT project. From the size of the box, we can see that the majority of metrics exhibits rather stable prediction performance. The best performing metrics achieve GM value above 0.7. Figure 3 presents the results obtained from the releases of PDE project. Comparing to the JDT project, the size of the box is wider and, hence, the performance of metrics is less stable. However, the best performing metrics achieve admirable GM values above 0.8.

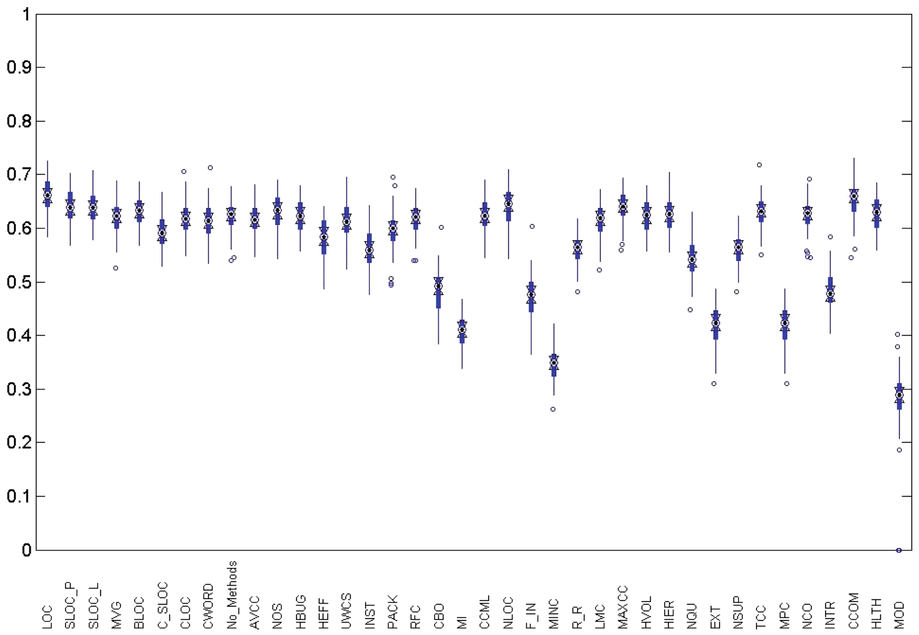

Fig. 2. Box and whisker plot for significant metrics in JDT

The threshold values of metrics that are significant in terms of statistical test ($p\text{-}value < 0.05$) and prediction performance ($GM > 0.6$) are given in Tables 4 and 5. We presented the mean and the standard deviation (in brackets) of threshold values for each release. The metrics are placed in rows and the releases are placed in columns. The threshold values of metrics 'HEFF' and 'HVOL' in Table 5 need to be multiplied by 10^4. In both projects combined, 21 different metrics were significant for defect prediction. There are 13 significant

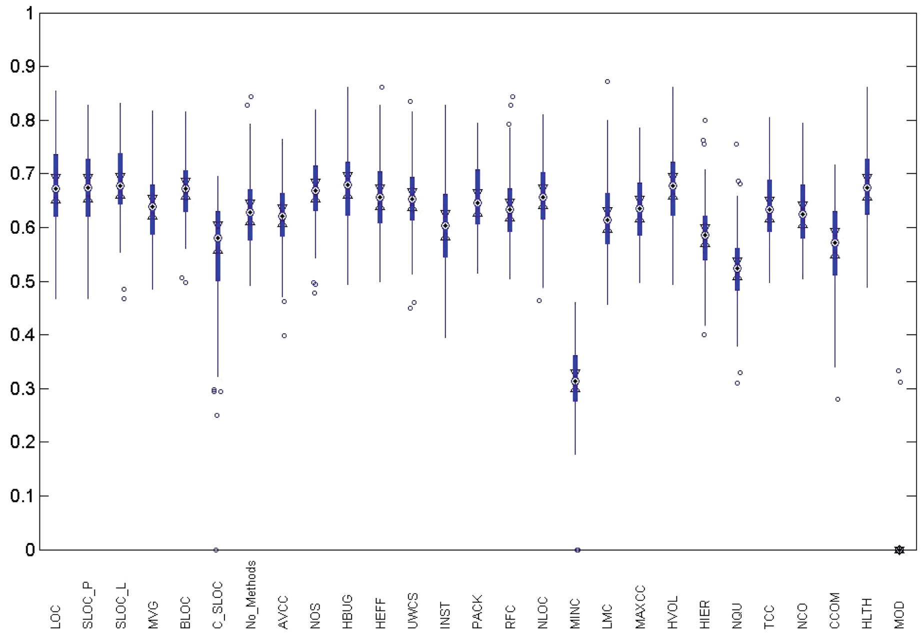

Fig. 3. Box and whisker plot for significant metrics in PDE

metrics in JDT project and 16 significant metrics in PDE project. The following 8 out of 21 metrics were significant for both JDT and PDE:

– ‘LOC’ - lines of code,
– ‘SLOC-P’ - physical executable source lines of code
– ‘SLOC-L’ - logical source lines of code
– ‘BLOC’ - blank lines of code
– ‘NOS’ - total number of Java statements in class
– ‘MAXCC’ - maximum cyclomatic complexity of any method in the class
– ‘TCC’ - total cyclomatic complexity of all the methods in the class
– ‘HLTH’ - cumulative Halstead length of all the components in the class

The level of imbalance is expressed as the percentage of faulty software modules and it is shown below the corresponding release label in Table 5. We ordered the releases in the table according to the level of imbalance to get an insight into the effect of data imbalance on threshold values. The standard deviation of each threshold is at least two orders of magnitude lower than its mean value. Hence, we conclude that the threshold values are very stable within the 10 folds of each release.

4.1 Discussion

In this paper we wanted to analyze the impact of data imbalance on the stability of threshold values. For the JDT project, all the thresholds have the lowest

Table 4. Threshold values for significant metrics (ordered by level of imbalance)

Project:	JDT						
Release:	2.0.	3.0.	3.2.	3.1.	2.1.	3.3.	3.4.
Imbalance:	46%	38%	36%	33%	32%	24%	18%
'LOC'	160.50 (0.95)	199.08 (2.03)	208.10 (1.29)	200.68 (1.71)	202.59 (1.77)	191.19 (1.36)	201.32 (1.53)
'SLOC_P'	100.84 (0.98)	123.23 (1.45)	132.70 (1.31)	127.80 (1.24)	129.40 (1.28)	118.00 (1.18)	122.79 (1.05)
'SLOC_L'	75.21 (0.73)	91.92 (1.02)	99.20 (1.10)	96.14 (0.92)	96.79 (0.98)	88.11 (0.86)	91.89 (0.77)
'BLOC'	18.51 (0.16)	21.50 (0.19)	24.16 (0.15)	23.37 (0.14)	23.12 (0.26)	20.73 (0.13)	22.60 (0.13)
'No_Methods'	9.01 (0.06)	10.34 (0.14)	10.80 (0.11)	10.61 (0.09)	10.57 (0.12)	10.00 (0.10)	9.94 (0.07)
'NOS'	61.84 (0.63)	74.16 (1.03)	78.67 (0.90)	75.65 (0.89)	77.65 (0.84)	71.56 (0.78)	71.99 (0.60)
'CCML'	39.22 (0.29)	48.83 (0.57)	45.05 (0.52)	43.07 (0.55)	44.80 (0.48)	46.70 (0.41)	48.95 (0.75)
'NLOC'	83.45 (0.75)	99.01 (1.41)	104.66 (1.13)	100.16 (1.18)	103.65 (1.04)	95.61 (0.89)	96.56 (0.75)
'MAXCC'	5.00 (0.04)	5.55 (0.07)	5.67 (0.04)	5.21 (0.03)	5.66 (0.04)	5.41 (0.03)	5.56 (0.03)
'TCC'	20.41 (0.19)	24.26 (0.35)	25.67 (0.23)	24.58 (0.27)	25.26 (0.23)	23.36 (0.24)	23.47 (0.17)
'NCO'	6.35 (0.05)	7.50 (0.12)	7.67 (0.10)	7.24 (0.07)	7.58 (0.10)	7.30 (0.08)	7.20 (0.07)
'CCOM'	14.67 (0.14)	16.74 (0.31)	16.27 (0.24)	15.96 (0.17)	16.93 (0.18)	16.87 (0.15)	16.55 (0.24)
'HLTH'	471.22 (5.37)	570.77 (8.74)	612.52 (6.13)	583.54 (6.66)	596.74 (6.47)	547.53 (6.43)	557.38 (5.04)

Table 5. Threshold values for significant metrics (ordered by level of imbalance)

Project:	PDE						
Release:	3.2.	3.3.	3.1.	3.0.	3.4.	2.0.	2.1.
Imbalance:	46%	44%	32%	31%	28%	19%	16%
'LOC'	143.16 (1.61)	146.49 (1.31)	154.70 (1.41)	151.88 (1.90)	141.68 (1.46)	147.51 (2.45)	165.59 (2.55)
'SLOC_P'	100.93 (1.29)	106.67 (1.13)	116.54 (1.27)	113.39 (1.59)	102.29 (1.23)	116.43 (2.22)	124.41 (1.99)
'SLOC_L'	73.27 (0.95)	78.37 (0.86)	84.05 (0.92)	81.55 (1.10)	75.72 (0.91)	85.88 (1.57)	89.29 (1.44)
'MVG'	15.09 (0.21)	16.47 (0.19)	16.88 (0.23)	16.51 (0.40)	15.76 (0.22)	15.62 (0.28)	17.39 (0.37)
'BLOC'	13.98 (0.12)	14.76 (0.12)	14.06 (0.14)	14.40 (0.15)	15.20 (0.13)	12.25 (0.22)	15.14 (0.18)
'NOS'	62.35 (0.73)	64.99 (0.72)	68.76 (0.83)	67.60 (0.86)	63.09 (0.75)	69.93 (1.26)	73.21 (1.38)
'HBUG'	0.80 (0.01)	0.84 (0.01)	0.89 (0.01)	0.86 (0.01)	0.82 (0.01)	0.87 (0.02)	0.93 (0.02)
'HEFF' [E+4]	2.78 (0.05)	2.99 (0.05)	3.01 (0.05)	2.93 (0.06)	2.98 (0.05)	2.78 (0.07)	3.09 (0.09)
'UWCS'	14.09 (0.15)	14.15 (0.12)	15.18 (0.17)	15.10 (0.20)	13.69 (0.16)	16.09 (0.18)	15.81 (0.23)
'PACK'	6.97 (0.07)	8.09 (0.06)	8.06 (0.07)	7.23 (0.08)	8.52 (0.08)	8.44 (0.11)	8.38 (0.11)
'RFC'	9.32 (0.10)	9.71 (0.07)	9.86 (0.11)	9.74 (0.09)	9.44 (0.08)	10.01 (0.13)	9.82 (0.09)
'NLOC'	85.34 (1.02)	88.03 (0.97)	95.47 (1.12)	93.38 (1.22)	85.03 (0.93)	95.49 (1.75)	101.91 (1.93)
'MAXCC'	4.41 (0.04)	4.63 (0.05)	4.53 (0.05)	4.49 (0.08)	4.55 (0.03)	4.25 (0.06)	4.57 (0.05)
'HVOL' [E+4]	2.41 (0.03)	2.52 (0.03)	2.66 (0.03)	2.57 (0.04)	2.45 (0.03)	2.61 (0.05)	2.80 (0.07)
'TCC'	19.24 (0.23)	20.53 (0.21)	20.62 (0.26)	20.19 (0.32)	20.04 (0.22)	19.52 (0.32)	20.99 (0.34)
'HLTH'	476.37 (6.10)	497.55 (5.54)	526.96 (6.21)	510.47 (7.13)	484.96 (5.97)	520.08 (10.17)	552.55 (11.50)

values in the most balanced dataset (release 2.0.) and rather stable value in the remaining datasets. Considering that JDT 2.0. is the earliest release and its liking rate is the lowest, the more emphasized difference may not be caused by data imbalance. For the PDE project, almost all the thresholds have the lowest values in the most balanced dataset (release 3.2.). Unlike the JDT 2.0., PDE 3.2. release is not the earliest nor does it have the lowest linking rate. Moreover, some metrics in PDE project exhibit a predominantly increasing rate of threshold values with higher levels of imbalance (for example 'LOC', 'SLOC-P', 'HLTH'). That is why we suspect that the data imbalance may have an impact on the threshold values for some metrics, after all. On the other hand, some metrics have a rather stable threshold value regardless of imbalance ('No_methods' 'MAXCC', 'NCO' and 'CCOM' for JDT and 'HBUG', 'UWCS', 'MAXCC', 'RFC' and 'TCC' for PDE).

Some of the metrics that are significant in both projects do have similar threshold values. For example, the range of threshold values for 'NOS' is [61–79] for JDT and [62–74] for PDE; for 'SLOC-P' it is [100–132] for JDT and [100–125] for PDE. On the other hand, some of the metrics are more diverse between the two projects which we have analyzed. For example, the maximum threshold value for 'LOC' is 202 for JDT and only 165 for PDE; for 'HTLT' it is 612 for JDT and 552 for PDE.

Our metrics 'No_Methods' and 'NSUB' are also known as 'WMC' and 'NOC' in related studies. The mean of threshold value of 'WMC' in related research varies from 10.49 [16], 13.5–13.94 [17] up to 23.17 [7]. The mean threshold value of 'No_Methods' varies from 9 to 10.8 in JDT and from 9.18 to 9.88 in PDE, which is similar to [16]. Although this metric passes the statistical significance criterion in every dataset, its GM value was lower than 0.6 for some datasets in PDE. Another metric that is often significant in related studies is 'CBO'. We obtained the same results for JDT, but the completely opposite results for PDE, where it was never significant. On the other hand, 'RFC' was found statistically significant both in our case study and in related research. However, the threshold values were 9.2–11.02 for JDT, 9.3–10.01 for PDE in our case study and 26.82 [16], around 40 [17] or 41.77 [7] in related studies. The 'DIT' metric was rarely significant in related studies and our case study confirmed this.

All of these examples confirm that general threshold values are hard to find and cross-project SDP is still difficult to achieve. Instead, we may only find generally significant or insignificant metrics. This case study also shows somewhat unexpected results. Software metrics that present very important features of object oriented code and its design, like for example cohesion or coupling, seam to be less often significant to defect prediction. This shows that most of previous research intentions that focused on a smaller number of intuitively important metrics offered a narrow view of this complex problem. That is why we believe this methodology should to be used to gain additional knowledge that can be used to improve the multivariate defect prediction models.

5 Conclusion and Future Work

This paper replicated the methodology for deriving the threshold values of software metrics which are effective for SDP. Afterwards, we analyzed the rate of metrics' significance, the spread of threshold values and we compared the central tendency of these values among different datasets. Unlike previous studies, our focus was not on finding generally applicable threshold values. Instead, we analyzed the stability of thresholds for within-project defect prediction. For that purpose we tested the metrics using a statistical test of significance and cross-validated the geometric mean accuracy of defect prediction. The results revealed that all the metrics that passed the two significance criteria had stable threshold values within the same release of a project. Data imbalance may have influenced the threshold values of some metrics, for which we noticed different values in the most balanced datasets and an increasing trend of values. Some of the derived thresholds were stable regardless of data imbalance and some were stable even across different projects. However, the results and the discussion pointed out that practitioners need to use the presented methodology for their specific projects because it is difficult to obtain general conclusions with regard to threshold values.

It is important to discuss the threats to construct, internal and external validity of this empirical case study. Construct validity is threaten by the fact that we do not posses industrial data. However, we used carefully collected datasets from large, complex and long lasting open source projects. The choice of threshold derivation model is a threat to internal validity, so we have used a model that was used and replicated in related studies. The main disadvantage of this case study is the inability to discover strong evidence regarding the influence of data imbalance on the stability of derived threshold values. Finally, the external validity is very limited by the diversity of data we have used. The results of this case study are reflecting the context of open source projects, in particular from the Eclipse community. We used only two projects and both are quite central for the Eclipse community. Our future work will expand the number of datasets and combine the results of this case study with our previous research in evolving co-evolutionary multi-objective genetic programming (CoMOGP) classification algorithm for SDP. We plan to use the calculated rate of significance for better definition of the weights that are assigned to each metric in the CoMOGP. We also plan to use the estimated threshold values for building the decision tree in the CoMOGP. Instead of taking the absolute value of each metric, we will use the distance of its value from the threshold. That way, we hope to achieve a more focused CoMOGP and improve the prediction results.

Acknowledgments. This work is supported in part by Croatian Science Foundation's funding of the project UIP-2014-09-7945 and by the University of Rijeka Research Grant 13.09.2.2.16.

References

1. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. **20**(6), 476–493 (1994)
2. Briand, L.C., Daly, J.W., Wust, J.K.: A unified framework for coupling measurement in object-oriented systems. IEEE Trans. Softw. Eng. **25**(1), 91–121 (1999)
3. Basili, V.R., Briand, L.C., Melo, W.L.: A validation of object-oriented design metrics as quality indicators. IEEE Trans. Softw. Eng. **22**(10), 751–761 (1996)
4. Radjenović, D., Heričko, M., Torkar, R., Živković, A.: Software fault prediction metrics: a systematic literature review. Inf. Softw. Technol. **55**(8), 1397–1418 (2013)
5. Arisholm, E., Briand, L.C., Johannessen, E.B.: A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. J. Syst. Softw. **83**(1), 2–17 (2010)
6. Shatnawi, R., Li, W.: The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. J. Syst. Softw. **81**(11), 1868–1882 (2008)

7. Shatnawi, R.: A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems. IEEE Trans. Softw. Eng. **36**(2), 216–225 (2010)
8. Galinac Grbac, T., Huljenić, D.: On the probability distribution of faults in complex software systems. Inf. Softw. Technol. **58**, 250–258 (2015)
9. Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S.: A systematic literature review on fault prediction performance in software engineering. IEEE Trans. Softw. Eng. **38**(6), 1276–1304 (2012)
10. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Trans. Knowl. Data Eng. **21**(9), 1263–1284 (2009)
11. Galinac Grbac, T., Runeson, P., Huljenić, D.: A second replicated quantitative analysis of fault distributions in complex software systems. IEEE Trans. Softw. Eng. **39**(4), 462–476 (2013)
12. Mauša, G., Galinac Grbac, T.: Co-evolutionary multi-population genetic programming for classification in software defect prediction: an empirical case study. Appl. Soft Comput. **55**, 331–351 (2017)
13. Graning, L., Jin, Y., Sendhoff, B.: Generalization improvement in multi-objective learning. In: The 2006 IEEE International Joint Conference on Neural Network Proceedings, pp. 4839–4846 (2006)
14. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Heidelberg (2003)
15. Martin, W.N., Lienig, J., Cohoon, J.P.: Island (migration) models: evolutionary algorithms based on punctuated equilibria. Handb. Evol. Comput. **6**, 1–15 (1997)
16. Arar, O.F., Ayan, K.: Deriving thresholds of software metrics to predict faults on open source software. Expert Syst. Appl. **61**(1), 106–121 (2016)
17. Shatnawi, R.: Deriving metrics thresholds using log transformation. J. Softw.: Evol. Process **27**(2), 95–113 (2015). JSME-14-0025.R2
18. Mauša, G., Galinac Grbac, T., Dalbelo Bašić, B.: A systemathic data collection procedure for software defect prediction. Comput. Sci. Inf. Syst. **13**(1), 173–197 (2016)
19. Oliveira, P., Valente, M.T., Lima, F.P.: Extracting relative thresholds for source code metrics. In: Proceedings of CSMR-WCRE, pp. 254–263 (2014)
20. Weiss, G.M.: Mining with rarity: a unifying framework. SIGKDD Explor. Newsl. **6**(1), 7–19 (2004)
21. Andersson, C., Runeson, P.: A replicated quantitative analysis of fault distributions in complex software systems. IEEE Trans. Softw. Eng. **33**(5), 273–286 (2007)
22. Fenton, N.E., Ohlsson, N.: Quantitative analysis of faults and failures in a complex software system. IEEE Trans. Softw. Eng. **26**(8), 797–814 (2000)
23. Bhowan, U., Johnston, M., Zhang, M., Yao, X.: Evolving diverse ensembles using genetic programming for classification with unbalanced data. IEEE Trans. Evol. Comput. **17**(3), 368–386 (2013)
24. Mauša, G., Galinac Grbac, T., Dalbelo Bašić, B.: Software defect prediction with bug-code analyzer - a data collection tool demo. In: Proceedings of SoftCOM 2014 (2014)
25. Mauša, G., Perković, P., Galinac Grbac, T., Štajduhar, I.: Techniques for bug-code linking. In: Proceedings of SQAMIA 2014, pp. 47–55 (2014)
26. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference and Prediction, 2nd edn. Springer, Heidelberg (2009)
27. Zimmermann, T., Nagappan, N.: Predicting defects using network analysis on dependency graphs. In: Proceedings of the 30th International Conference on Software Engineering. ICSE 2008, pp. 531–540. ACM, New York (2008)

28. Bender, R.: Quantitative risk assessment in epidemiological studies investigating threshold effects. Biometrical J. **41**(3), 305–319 (1999)
29. Bhowan, U., Johnston, M., Zhang, M., Yao, X.: Reusing genetic programming for ensemble selection in classification of unbalanced data. IEEE Trans. Evol. Comput. **18**, 893–908 (2013)