An empirical study on the influence of context in computing thresholds for Chidamber and Kemerer metrics

Leonardo C. Santos, Renata Saraiva, Mirko Perkusich, Hyggo O. Almeida and Angelo Perkusich
Federal University of Campina Grande, Campina Grande, Brazil
{leonardo.santos, renata.saraiva, mirko.perkusich, hyggo, perkusic}@embedded.ufcg.edu.br

Abstract

Software metrics have a fundamental role in the process of software quality management. However, in most cases, they are only used to quantify attributes, not supporting decision-making during the software life cycle. To support decision-making, it is necessary to give them by defining thresholds. In the literature, several approaches have been proposed with this purpose. On the other hand, most of them do not consider context factors such as the domain. Given this, in this paper, we evaluate if context factors influence the definition of thresholds for software metrics. Our work is restricted to Chidamber and Kemerer metrics, due to availability of data. We conducted an empirical study composed of two quasi-experiments. Each quasi-experiment uses an approach presented in the literature to define thresholds for software metrics, with the defined thresholds as the dependent variable. As the factor, we used a variable with two possible treatments: to consider the context or not. To define context, we used factors presented in the literature. As the objects of study, we used the source code of fifteen Java-based open-source projects. For measurement purposes, we used the six original Chidamber and Kemerer metrics. For both quasi-experiments, the accuracy of the definition of thresholds improved by considering the context. Therefore, we concluded that context factors influence the definition of the threshold for Chidamber and Kemerer metrics, which is an indicator that it influences other software metrics.

Software measurement; thresholds derivation; CK metrics.

1. Introduction

Software metrics is a collective term used to describe the wide range of activities concerning measurement in software engineering [1]. The reasons to use software metrics are: (i) to assist in project planning; (ii) to determine the strengths and weaknesses of the process and the product; and (iii) to evaluate the impact of a used particular technique. In practice, the use of metrics in large organizations such as HP, Motorola and NASA was evaluated by Ordonez and Haddad [2]. The result of this study indicated that metrics, when used early in the software development cycle, help to correct requirement failures and prevent errors.

Despite the potential benefits of metrics, in most cases, they are only used to quantify attributes and do not support decision-making [3]. According to Ferreira *et al.* [4], this has been identified as one of the reasons why metrics are not effectively used in industry. To maximize the use of metrics and support their interpretation, it is essential to define significant thresholds.

Many approaches to identify thresholds have been proposed in the literature. Ferreira *et al.* [4] used the *EasyFit* tool [5] to find the distribution most similar to the distribution of the metric. From percentile cuts, the categorization of thresholds was defined as good, moderate and bad. Foucault *et al.* [6] and Alves *et al.* [3] also used percentile cuts to compute thresholds. Foucault *et al.* [6] used a statistical analysis called *bootstrap* [7] to estimate the confidence interval of the percentile.

Receiver Operating Characteristic (ROC) curves were used by Shatnawi *et al.* [8] to associate metric to errors and find the threshold that provides better justification for these errors. In 2015, Shatnawi [9] proposed an approach that performs a logarithmic transformation on the data to reduce the skewness. In this approach, thresholds were extracted from the mean and standard deviation of the distribution related to the metric. Most of the works, such as Ferreira *et al.* [4], Alves *et al.* [3] and Shatnawi *et al.* [8], did not consider context variables to define the thresholds. On the other hand, Zhang *et al.* [10], in an empirical study that collected data from 320 software systems, demonstrated that the distribution of maintainability metrics values are influenced by context variables such as the application domain, the programming language and the number of changes made during the software development, giving indications that thresholds can also be influenced by these variables.

In this study, we hypothesize that context variables influence the software metrics' threshold, given that the distri-

DOI reference number: 10.18293/SEKE2017-044

bution of a metric influences its interpretation (i.e., thresholds). In this paper, we investigate if context variables influence the accuracy of the definition of software metrics' thresholds. With this purpose, we conducted an empirical study composed of two quasi-experiments. Each quasi-experiment uses a solution proposed in the literature to define thresholds for software metrics, with the calculated thresholds as the dependent variable.

Based on the number of referrals, we used the approaches proposed by Foucault *et al.* [6] and by Shatnawi [9]. As the factor of the study, we used a variable with two possible treatments: to consider the context or not. As the objects of study, we used the source code of fifteen Java-based open-source projects. For measurement purposes, we used the six original Chidamber and Kemerer (CK) metrics [11].

The remainder of this paper is organized as follows: in Section 2, we present the background regarding software metrics and thresholds derivation. In Section 3, we describe the empirical study design. In Section 4, we present the analysis and results. In Section 5, we present the threats to validity. Finally, in Section 6, we present our conclusions and future work.

2. Background

2.1. Chidamber and Kemerer Metrics

Metrics are used as a control instrument in the software development and maintenance process. Since 1970, hundreds of metrics have been proposed in the literature [12]. For instance, Chidamber and Kemerer [11] proposed a set of object-oriented metrics that is widely used by researchers, which are presented as follows:

- *Coupling Between Object classes (CBO)*: the coupling of a class is characterized by the number of relationships that a class has (i. e. counted for method calls, field accesses, inheritance, method arguments, return types, and exceptions). High coupling indicates a low reuse and a rise on the sensitivity to changes;
- *Depth of Inheritance Tree (DIT):* this metric calculates the depth of the inheritance tree (i.e., the distance between a class and its root class). It indicates that the deeper a class is in a class hierarchy, the more methods are inherited. Thus, the class becomes more complex and prone to errors;
- *Number Of Children (NOC):* represents the number of subclasses that inherit characteristics from a given class. This information provides evidence of the importance of the class importance to the project. A high value for this metric may correspond to inappropriate abstractions or mistakes related to inheritance concepts;
- *Response For a Class (RFC):* refers to the number of methods that can be executed by a class instance in response to an event or a received message. The higher the value, the greater the complexity of its testing and maintenance;
- *Lack of Cohesion in Methods (LCOM):* refers to the number of pairs of methods of a particular class in which the similarity is zero, minus the number of pairs of methods in which the similarity is nonzero. The similarity is calculated by the common use of variables of a class instance. Thus, a high value means that the class is not cohesive.
- *Weighted Methods per Class (WMC):* refers of the sum of the complexities of the methods of a given class. The higher the value, the more complex the class is.

Many studies have verified the relationship between CK metrics and faults in classes. In a systematic review, Jureczko and Madeyski [13] analyzed their effects on fault proneness. They showed that object-oriented metrics are better at finding fault than procedural metrics. Furthermore, they claim that CK metrics form the most common set of metrics to predict failures in classes. In Table 1, we detail the effect of CK metrics in a class' fault-proneness as presented in Shatnawi [9]. The first column corresponds to CK metrics. Second and third columns correspond to the quantity of research papers that present the negative and positive impact of CK metrics in a class' fault-proneness. Finally, the last column corresponds to the research papers that present data refuting the hypothesis that there is a relationship between CK metrics and faults.

Table 1: A summary of the CK metrics' impact in fault proneness [13, 9].

Metric	Positive	Negative	Not significant
WMC	12	0	0
DIT	4	2	6
NOC	2	3	4
CBO	11	0	1
RFC	11	0	0
LCOM	6	0	1

2.2. Thresholds Definition

The effective use of software metrics is hampered by the lack of significant thresholds [3]. In the literature, few metrics have defined thresholds. Furthermore, many researchers have proposed different approaches to define them [3, 4, 6, 14, 15, 9, 16, 8, 10].

Alves *et al.* [3] present a method that determines threshold empirically from measurement data. (i.e., benchmarking). The method is based on statistical properties of the

metric such as scale and distribution. To evaluate their approach, they collected data from 100 object-oriented software systems to calculate thresholds, which were successfully used to assist on software analysis, benchmarking and certification. The main risk of such a solution is to use thresholds to assist decision-making that were calculated for a different context.

In Sánchez-González *et al.* [16], an empirical study was performed to evaluate the effectiveness of two threshold definition techniques: ROC curves [8] and the Bender method [17] to define thresholds. As objects of study, they used measures for business process models. They concluded that ROC curves obtain more accurate thresholds.

In the works of Oliveira *et al.* [15, 14], the concept of relative thresholds is proposed as well as a tool for extracting these thresholds. Their approach handles the heavy-tailored distribution of source code metrics by complementing absolute thresholds with a percentage of software code entities that must follow it. The technique is validated with an industrial case study. As Alves *et al.* [3], its limitation is that the calculated threshold and percentage might be dependent on the context.

In Foucault *et al.* [6], a solution based on statistical methods was presented. This approach is based on (i) *double sampling* [18] to randomly selects projects samples; and (ii) *bootstrap* to estimate the thresholds based on quartiles. Despite the potential of this approach, the validation process was limited to a test to identify the best configuration for the approach itself since, according to the authors, the two statistical methods are widely used.

In Shatnawi [9], a solution based on logarithmic transformation was presented. In this approach, initially, the data is transformed using the natural log, leaving the symmetric data thus closer to a normal distribution. Afterward, a temporary reference value (T') is collected using the mean (M) and standard deviation (SD) so that $T' = M + SD$ or $T' = M - SD$. Finally, the T' is converted to the original distribution by using the exponent function of T', generating the final reference value.

3. Study Design

To evaluate if context factors influence the definition of software quality metrics' thresholds, we performed an empirical study composed of two quasi-experiments. Each quasi-experiment used one solution to define the thresholds of software metrics presented in the literature. The solutions used are the ones presented by Shatnawi [9] and Foucault *et al.* [6]. The defined thresholds are the dependent variable. As the factor, we used a variable with two possible treatments: to consider the context or not. The context factors were defined according to Zhang *et al.* [10]. As database, we used six CK metrics extracted of the source code of fifteen Java-based open-source projects.

3.1. Scope

The goal of the study is to evaluate if context factors influence the definition of software quality metrics' threshold in the context of Chidamber and Kemerer metrics. Therefore, we addressed the following research question:

RQ: Does considering context factors improve the quality of the definition of software quality metrics' thresholds?

Given the research question, we defined the following informal hypotheses:

H0: The results are the same or worse.

HA: The results are better.

3.2. Objects of study

For both quasi-experiments, we used the same data as Shatnawi [9], which is composed of several releases of fifteen open-source projects written in the Java programming language. We classified each project according to its application domain and the number of changes given the definitions presented in Zhang *et al.* [10]. The application domain of 73% of the projects is software development, 13% is the development of build tools, and 13% is the development of frameworks. For the number of changes, we used the thresholds defined by Zhang *et al.* [10] given the number of commits to the repository: 12 as *very small*; 123 as *small*; 413 as *medium*; 1142 as *large*; and 94853 as *extra large*.

3.3. Variables and treatment

In this study, we have two independent variables: (i) the technique used to define the thresholds and (ii) the context usage. As already stated, for the first variable (i), we used two options: the solutions presented in Shatnawi [9] and Foucault *et al.* [6]. In this study, we executed one quasi-experiment for each possible value of the first variable (i). For the second variable (ii), there are two options: to consider the context or not. To define the context, we used two out of three variables identified by Zhang *et al.* [10]: application domain and the number of changes, since the programming language of the base used is the same. Therefore, for each quasi-experiment, the treatment factor is the *context usage*. Furthermore, for each quasi-experiment, we have one dependent (*i.e.*, response) variable: the quality of the thresholds defined.

3.4. Measurement procedure

To evaluate the hypotheses, assess the research questions and research goals, we collected metrics. Furthermore, by defining the metrics, we formalized the hypotheses presented in Section 3.1 statistically test them. As stated in Section 3.3, for each quasi-experiment, we have one dependent variable: the quality of the thresholds defined. They were collected through the open source tool ckjm[1] and pub-

[1] http://www.spinellis.gr/sw/ckjm/

lished by the Metrics Repository [19, 20]. The measurement procedure was based on Shatnawi [9].

In this study, we assumed that the highest the source code quality, the fewer faults the software has. Since all of the objects of study are software written in Java code (*i.e.*, Object-oriented), as presented in 3.2, in order to measure the source code quality, we used the most popular Object-oriented metric set: CK metrics [11], which we presented in an overview in Section 2.1.

To measure the number of software faults, we used the BugInfo tool[2]. With this tool, we collected the number of faults of each class for all of the versions of the projects. This tool analyses the commits of a given project and detects, using regular expressions, if given classes had faults. Therefore, whenever a commit message is compatible with a defined regular expression, the number of faults for the given classes is incremented. Given this, we used the number of faults detected using the BugInfo tool to indicate source code quality.

To measure the threshold quality, we used the F-measure. For this purpose, the first step was to group the objects of study given the context variables presented in Section 3.3. Afterward, we separated the group into two samples. The first sample, called *context sample*, was composed of data from projects with the given characteristics: developed in *Java*, with *software development* as the application domain and classified as *extra large* regarding the number of changes. The second sample, called *control sample*, is composed of the remaining data.

For each quasi-experiment, we used the samples to compare the results found by applying the given thresholds' definition solution (i.e., Shatnawi [9] or Foucault *et al.* [6]) considering or not context variables. For each sample, we applied the k-fold cross-validation with $k = 10$. For every iteration, we used 9k to define the thresholds for six CK metrics: CBO, DIT, NOC, RFC, LCOM and WCM. We used 1k interaction' to evaluate the thresholds. As a result, we have two possible outcomes: (i) faulty classes, if $M \geq R$; or (ii) nonfaulty classes, if $M < R$, where M is the collected metric value and R is the defined threshold. The procedure is presented in Figure 1.

Table 2: The confusion matrix based on a threshold value.

Predicted	**Faulty**	**Nonfaulty**
M >= R	True positive	False positive
M < R	False negative	True negative

We used the confusion matrix presented in Table 2 to measure the performance of using the thresholds model in identifying actual fault classes using three measures: *Recall*, *Precision*, and *F-measure*. These measures are calculated as follows:

[2]https://kenai.com/projects/buginfo

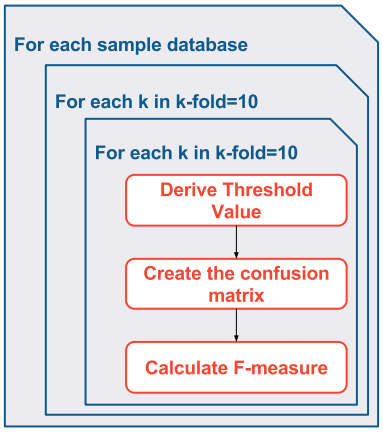

Figure 1: Design.

$$Recall = \frac{TP}{TP + FN} \quad (1)$$

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$F - measure = \frac{(\beta^2 + 1) * Precision * Recall}{\beta^2 * Precision + Recall} \quad (3)$$

- The term β is used to assign a weight to *Recall*. In our work, β is equal to 1, and *Recall* and *Precision* are equally weighted.
- True Positive (TP): faulty classes that are correctly classified as such (*i.e.* there are faults fixed in the class and the metric value exceeds the threshold).
- False Negative (FN): faulty classes that are misclassified as nonfaulty (*i.e.* there are faults fixed in the class, but the metric value is less than the threshold).
- True Negatives (TN): nonfaulty classes that are correctly classified as such (*i.e.* there are no faults fixed in the class and the metric value is less than the threshold).
- False Positives (FP): nonfaulty classes that are misclassified as faulty (*i.e.* there are no faults fixed in the class, but the metric value exceeds the threshold).

The values of both *Recall* and *Precision* are between $[0, 1]$. Values that are closer to 1 mean better results, with 1 as an ideal value (*i.e.*, without FN or FP). According to Shatnawi [9], in practice, it is hard to achieve high *Recall* and high *Precision*. Finally, we used the *F-measure* to evaluate the overall performance of classification, combining *Recall* and *Precision*.

At the end, for each metric, we had a set of *F-measures* for the context sample and a different set for the control sample. We compared the pairs of *F-measures* of both sets using the non-parametric Wilcoxon test [21]. Thus, for each quasi-experiment, we formally defined a null and alternative hypothesis:

Therefore, for each quasi-experiment, we formally defined a null and alternative hypothesis:

H0: $\Omega \leq \Psi$, where Ω is the F-measure for the *context sample* and Ψ is the F-measure for the *control sample*.

HA: $\Omega > \Psi$.

4. Analysis and Results

4.1. First quasi-experiment: Foucault *et al.* approach

For the first quasi-experiment, we evaluated the solution proposed by Foucault *et al.* [6]. This approach, as presented in Section 2.2, is based on quantile analysis. We decided to use only the first percentile (80%) since it has the ability to represent the whole. An average of the thresholds' values found can be seen in Table 3. By comparing the thresholds defined for both samples (*i.e.*, context and control), it is possible to see that the DIT metric was the same for both samples.

Table 3: An average of the thresholds' values found using Foucault *et al.*'s approach.

Metric	Context threshold	Control threshold
WMC	9.00	10.05
NOC	0.00	0.20
CBO	9.05	10.10
RFC	25.15	27.85
DIT	2.50	2.50
LCOM	33.30	44.64

In Table 4, we present the average *F-measure* of the two data samples. Furthermore, we present the improvement of the threshold definition, $\theta = \Omega/\Psi$, and the *p-value* resulting from the Wilcoxon test. By analyzing the results, we conclude that, for all metrics, the threshold definition was slightly improved by considering the context. Since for all metrics $p-value < 0.05$, we refute the null hypothesis that states that considering the context to define the thresholds does not influence its definition for the solution presented by Foucault *et al.* [6].

4.2. Second quasi-experiment: Shatnawi approach

For the second quasi-experiment, we evaluated the solution proposed by Shatnawi [9]. This approach, as presented in Section 2.2, is based on log transformation.

Initially, for each k-fold interaction, two thresholds derived from each of the six CK metrics, one for each of the two database samples. An average of the thresholds' values found can be seen in Table 5. By comparing the thresholds for both samples, the differences were minimal.

Table 4: Results for Foucault *et al.*'s approach.

Metric	F-measure		θ	*p-value*
	Context	Control		
WMC	0.23	0.20	15.00%	1.62e-04
NOC	0.42	0.18	133.30%	5.41e-06
CBO	0.23	0.14	64.30%	5.41e-06
RFC	0.22	0.19	15.80%	2.16e-05
DIT	0.18	0.10	80.00%	5.41e-06
LCOM	0.16	0.15	6.70%	1.43e-03

Table 5: An average of the thresholds' values found using Shatnawi's approach.

Metric	Context threshold	Control threshold
WMC	1.79	1.82
NOC	0.75	0.74
CBO	1.81	1.85
RFC	3.79	3.90
DIT	0.92	0.93
LCOM	0.62	0.62

In Table 6, we present the average *F-measure* of the two data samples. Furthermore, we present the improvement of the threshold definition, $\theta = \Omega/\Psi$, and the *p-value* resulting from the Wilcoxon test. By analyzing the results, we can notice that for all metrics, the threshold definition was slightly improved by considering the context. This is possible because the thresholds were defined for different samples. Since for all metrics $p - value < 0.05$, we refute the null hypothesis that states that considering the context to define the thresholds does not influence its definition for the solution presented by Shatnawi [9].

Table 6: Results for Shatnawi's approach.

Metric	F-measure		θ	*p-value*
	Context	Control		
WMC	0.40	0.31	29.03%	5.41e-06
NOC	0.09	0.06	50.00%	5.41e-06
CBO	0.39	0.30	30.00%	5.41e-06
RFC	0.39	0.31	25.81%	5.41e-06
DIT	0.42	0.32	31.25%	5.41e-06
LCOM	0.34	0.28	21.43%	9.08e-05

5. Threats to Validity

A threat to internal validity is that we did not consider the impact of many software development factors such as development process and team experience. In addition, the experiment would strengthen the evidence by performing

the same tests alternating the control and context groups, showing the threat of the control group. Finally, we only evaluated Java-based projects.

As threat to external validity, we only used data from fifteen systems and two solutions to derive thresholds, which might not be enough to generalize the data in the context of CK metrics. Furthermore, we cannot generalize the results for other software metrics.

As threat to construct validity, we identified a faulty class through regular expressions in commit messages using the BugInfo tool, which might not be accurate. In addition, we used the percentage of faults to validate the thresholds. Even though there is evidence that the six CK metrics can indicate flaws in software [13] and is the same process used by Shatnawi [9], the reliability of the metric can be a threat to the validity of our study.

6. Conclusions

In this paper, we presented results of an empirical study performed to evaluate the influence of the context on the definition of software metrics' thresholds. The scope of our contribution is restricted to CK metrics. We executed two quasi-experiments, each one using one solution proposed in the literature to define thresholds for software metrics. We used the solutions presented in Foucault *et al.* [6] and Shatnawi [9]. The objects of study were data from fifteen open-source Java-based projects.

We used the Wilcoxon test to evaluate both quasi-experiments. For both cases, the threshold's definition accuracy improved by considering the context to compute it. Therefore, we present evidence that, in the context of CK metrics, it is relevant to consider the context to define thresholds. As a result, we conclude that thresholds can only be reused for projects with similar contexts.

In our future work, we plan to study the implications of our results in decision-making and software quality management. Furthermore, we plan to execute experiments to investigate how, for other types of software metrics, the context influences the definition of thresholds.

References

[1] N. E. Fenton and M. Neil, "Software metrics: roadmap," in *Proceedings of the Conference on the Future of Software Engineering*. ACM, 2000, pp. 357–370.

[2] M. J. Ordonez and H. M. Haddad, "The state of metrics in software industry," in *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on*. IEEE, 2008, pp. 453–458.

[3] T. L. Alves, C. Ypma, and J. Visser, "Deriving metric thresholds from benchmark data," in *2010 IEEE International Conference on Software Maintenance*. IEEE, sep 2010, pp. 1–10.

[4] K. A. Ferreira, M. A. Bigonha, R. S. Bigonha, L. F. Mendes, and H. C. Almeida, "Identifying thresholds for object-oriented software metrics," *Journal of Systems and Software*, no. 2, pp. 244–257, feb 2012.

[5] R. Martin, "Oo design quality metrics," *An analysis of dependencies*, vol. 12, pp. 151–170, 1994.

[6] M. Foucault, M. Palyart, J.-R. Falleri, and X. Blanc, "Computing contextual metric thresholds," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*. New York, New York, USA: ACM Press, mar 2014, pp. 1120–1125.

[7] B. Efron, "Bootstrap methods: another look at the jackknife," *The annals of Statistics*, pp. 1–26, 1979.

[8] R. Shatnawi, W. Li, J. Swain, and T. Newman, "Finding software metrics threshold values using ROC curves," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 22, no. 1, pp. 1–16, jan 2010.

[9] R. Shatnawi, "Deriving metrics thresholds using log transformation," *Journal of Software: Evolution and Process*, vol. 27, no. 2, pp. 95–113, feb 2015.

[10] F. Zhang, A. Mockus, Y. Zou, F. Khomh, and A. E. Hassan, "How Does Context Affect the Distribution of Software Maintainability Metrics?" in *2013 IEEE International Conference on Software Maintenance*. IEEE, sep 2013, pp. 350–359.

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

[12] Z. Bukhari, J. Yahaya, and A. Deraman, "Software metric selection methods: A review," in *Electrical Engineering and Informatics (ICEEI), 2015 International Conference on*. IEEE, 2015, pp. 433–438.

[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] P. Oliveira, F. P. Lima, M. T. Valente, and A. Serebrenik, "Rttool: A tool for extracting relative thresholds for source code metrics," in *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2014, pp. 629–632.

[15] P. Oliveira, M. T. Valente, and F. P. Lima, "Extracting relative thresholds for source code metrics," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, feb 2014, pp. 254–263.

[16] L. Sánchez-González, F. García, F. Ruiz, and J. Mendling, "A study of the effectiveness of two threshold definition techniques," in *Evaluation & Assessment in Software Engineering (EASE 2012), 16th International Conference on*. IET, 2012, pp. 197–205.

[17] R. Bender, "Quantitative risk assessment in epidemiological studies investigating threshold effects," *Biometrical Journal*, vol. 41, no. 3, pp. 305–319, 1999.

[18] S. K. Thompson, "Simple random sampling," *Sampling, Third Edition*, pp. 9–37, 2012.

[19] M. Jureczko, "Significance of different software metrics in defect prediction," *Software Engineering: An International Journal*, vol. 1, no. 1, pp. 86–95, 2011.

[20] L. Madeyski and M. Jureczko, "Which process metrics can significantly improve defect prediction models? an empirical study," *Software Quality Journal*, vol. 23, no. 3, pp. 393–422, 2015.

[21] D. F. Bauer, "Constructing confidence sets using rank statistics," *Journal of the American Statistical Association*, vol. 67, no. 339, pp. 687–690, 1972.