OBJECT-ORIENTED METRICS PREDICTION AS A TOOL FOR SOFTWARE QUALITY EVALUATION

Conference Paper · June 2023

CITATIONS
0

READS
70

3 authors:

Milica Tufegdžić
Academy of Professional Studies Sumadija
35 PUBLICATIONS **48** CITATIONS
SEE PROFILE

Vladimir Nedic
Academy of Professional Studies Sumadija
42 PUBLICATIONS **936** CITATIONS
SEE PROFILE

Aleksandar Mišković
Academy of professional studies Sumadija, Serbia
24 PUBLICATIONS **29** CITATIONS
SEE PROFILE

Quality Festival 2023
ISBN 978-86-6335-104-2

Milica Tufegdžić [1]
Vladimir Nedić
Aleksandar Mišković

OBJECT-ORIENTED METRICS PREDICTION AS A TOOL FOR SOFTWARE QUALITY EVALUATION

Abstract: *Chidamber & Kemerer metrics as a set of six basic metrics, in combination with Lines of Code, is used as a quantitative measure of object-oriented software characteristics. Guided by the fact that these measurements are not an easy task and time consuming, we propose an artificial neural network with optimal three-layer back-propagation for quick object-oriented metrics' prediction, based on measured values of Weighted Methods per Class and Lines of Code. The study is conducted at the sample of 367 cases of metrics' values taken for jedit 4.2. Performances in the form of correlation coefficient and mean square error are measured, as well as the performance during the learning process in terms of gradient, momentum and validation checks with respect to number of epochs. Obtained values indicate a significant and almost perfect adaption of outputs to the targets, creating an opportunity get all the values for basic metrics, with proper accuracy.*

Keywords: *Chidamber & Kemerer metrics, software assessment, artificial neural network*

1. Introduction

Quality attainment is the most critical issue in software development, especially in today's dynamic environment. Therefore, ensuring the software's commercial success implies conforming to a number of client requirements with zero error, under specified conditions. Meeting the internal and external set of features and characteristics should be achieved through continuous monitoring and software quality evaluation (Arvanitou et al., 2016). The most common way to do this is to conduct different software measurements with the aim to obtain numeric data related to software projects, and to determine factors such as project size, time spent, cost, etc. (Yücalar & Boranda, 2016). These measurement are useful in assessing software quality attributes and could be used in predicting software testability (Badri & Toure, 2012; Juliano et al., 2014), as well as evaluating maintainability, understandability, and reliability (Desai, 2014; Juliano et al., 2014).

Series of International standards ISO/IEC 25000 SQuaRE (Software Product Quality Requirement and Evaluation) has been developed with the aim to provide proper framework for evaluating software quality. Due the fact that SQuaRE is the result of evolution of several standards, it consists of five divisions: Quality Management (2500x), Quality Model (2501x), Quality Measurement (2502x), Quality Requirements (2503x) and Quality Evaluation (2504x). According to ISO/IEC 25000, evaluation could be done by

Corresponding author: Milica Tufegdžić
Email: mtufegdzic@asss.edu.rs

measuring internal or external properties, such are static measures or by measuring code's behavior during execution (Perdomo & Zapata, 2021; Norbert & András, 2021).

Different aspects of quality require different software metrics that are used to provide a quantative way to assess the quality, through objective reproducible measurements (Dubey et al., 2012). This resulted in development of a large number of metrics, as well as many tools for collecting metrics from software (Singla & Singh, 2014). These tools can be general or dependent on programming language, provided as stand-alone software or integrated, in the form of plugin (Molnar et al., 2019). Thereby, regardless of the type or form of these tools, it is necessary to know the context of the measured object for software evaluation (Dahab & Maag, 2019).

Object-oriented approach to software development, as an expansion of procedural approaches, separates data and control. Due the fact that the system is viewed as a collection of objects, object-oriented metrics must provide measurements that can be applied on classes and characteristics such as object abstraction techniques, encapsulation, inheritance, polymorphism, and information hiding. There are three categories of this metric, namely process metric, product metric, and project metric (Kumar & Vijayaraghavulu, 2021; Shaik et al., 2010). Different sets of object-oriented metrics are proposed: Chen metrics, Chidamber & Kemerer (CK) metrics, Matrin metrics, Lorenz and Kidd metrics, MOOSE (Metrics for Object-Oriented Software Engineering) metrics, MODD (Metrics for Object Oriented Design) metrics, Goal Question metrics, QMOOD (Quality Model for Object Oriented Design) metrics, Li and Henry metrics, and SATC (Software Assurance Technology Center) for object-oriented metrics (Desai, 2014; Dubey et al., 2012; Yücalar & Boranda, 2016). CK recommends six basic metrics: Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Object Classes (CBO), Response for a Class (RFC), and Lack of Cohesion in Methods (LCOM) (Chidamber & Kemerer, 1994; Desai, 2014; Juliano et al., 2014; Dubey et al., 2012; Molnar et al., 2019; Padhy et al., 2017; Sabhat et al., 2017; Ţirban, 2018; Yücalar & Boranda, 2016). Li and Henry took five metrics from CK (WMC, DIT, NOC, RFC, and LCOM), and added five more metrics: Message Passing Coupling (MPC), Data Abstraction Coupling (DAC), Number of Methods (NOM), Number of Semicolons (SIZE1), and Number of Properties (SIZE2) (Desai, 2014). Martin focuses at architectural design issues, defining six metrics (Relational cohesion (H), Afferent coupling (CA), Efferent coupling (CE), Abstractness (A), Instability (I), and Distance from main sequence (D)), with the aim to reduce dependencies between classes (Desai, 2014). MOOD metrics deals with basic object-oriented paradigms, covering six metrics, namely Method Hiding Factor (MHF), Attribute Hiding Factor (AHF), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), Polymorphism Factor (POF), and Coupling Factor (COF). MHF and AHF cover encapsulation, MIF and AIF cover inheritance, POF covers polymorphism, while COF deals with message passing (Desai, 2014; Yücalar & Boranda, 2016). QMOOD set of metric allows total quality index of the software calculation based on seven metrics (Lines of Code (LOC), Average Complexity (AC), Number of Instance Variables (NIV), WMC, DIT, RFC, LCOM) (Dubey et al., 2012; Yücalar & Boranda, 2016). Some traditional metrics includes Cyclomatic Complexity (CC) to measure algorithm complexity in the class method, and LOC (Dubey et al., 2012).

Different values for metric numbers have been proposed for requirement, design, and testing phases of software development

lifecycle, such are 27, 42, and 46 metrics, respectively (Singla & Singh, 2014). The fluctuation analysis of 19 object-oriented metrics, conducted at 20 open-source software projects shows that the source code metrics are more sensitive, compared to design metrics (Arvanitou et al., 2016). Object-oriented metric has proved to be successful with identifying classes that should be tested first (Yücalar & Boranda, 2016). Basic six CK metrics were combined into three metrics for software quality assessment: WMPRC (WMC + RFC), DITNC (DIT + NOC), and CBLCM (CBO + LCOM) (Padhy et al., 2017). Sofware CK metrics for Java and C++ projects that can indicate possible failures with high probability were analyzed (Juliano et al., 2014). Some automated tool for software code quality assessment based on inheritance, encapsulation, and polymorphism, was considered (Mwangi et al., 2014). WMC and LOC were considered as the class size factors with the aim to develop models for software defects prediction (Jureczko & Spinellis, 2010). There is a possibility to predict class fault-proneness using five out of six CK metrics (Basili et al., 1996).

Different open source and licensed tools were used for Geant4 software metrics. Some examples include CCCC (C and C++ Code Counter), CLOC (Count Lines of Code), Pmccabe, SLOCCount (Source Lines of Code Count), Unified Code Count, and Difftool (Ronchieri et al., 2016; Vijay, 2016). The metrics were aggregated in categories like Program Size, Code Distribution, Control Flow Complexity, and Object-Orientation (Ronchieri et al., 2016). Coupling, inheritance, cohesion, and structural complexity as internal characteristics were computed using VizzAnalyzer tool. The metrics used for coupling are extracted with CBO, DAC, and Message Pass Coupling (MPC). The inheritance is measured via DIT, NOC, LCOM, Improvement to Lack of Cohesion in Methods (ILCOM), and Tight Class Cohesion (TCC), while the metrics related to the structural complexity of classes is derived using Locality of Data (LD), Number of Attributes and Methods (NAM), NOM, RFC, and WMC. The metric related to code documentation include Length of Class Name (LEN) and Lack of Documentation (LOD), as well as LOC. The measurements were done at three open-source applications developed in Java, and the obtained values were analyzed using statistical tools, followed by establishing proper correlations between pairs of metrics (Molnar et al., 2019).

Ten different metrics were used for: complexity (WMC, DIT, RFC, and Average Method Complexity (AMC)), coupling (CBO, CA, and CE), and cohesion (two versions of LCOM and Cohesion Among Methods of a Class (CAM)) (Bos, 2019). System which calculates CK metrics, as well as hierarchical metrics, and project and module metrics, was deployed and used for Java application (Thirugnanam & Swathi, 2010). Web-based tool that enables standard software metrics and statistic calculations was developed (Vijay, 2016). Model for predicting bugs in object-oriented software using metrics was established (Kumar & Vijayaraghavulu, 2021). CK metrics related to the JUnit test cases, together with LOC, was used to evaluate the relationship between object-oriented metrics and unit testing effort, with the aid of multivariate logistic regression models (Badri & Toure, 2012). Statistical analysis was used for establishing correlation between reliability and object-oriented metrics. WMC, RFC and LOC show strong correlations with reliability, while CBO and LCOM have moderate correlations. NOC has weak correlation (Ţirban, 2018). The results of the study conducted at 10 software modules indicate that optimal metrics for object-oriented software are RFC, LOC and WMC,

while NOC and DIT are less important (Lamba et al., 2017).

Improving the software measurement process was done using unsupervised learning algorithm X-MEANS, with the aim to generate a reliable analysis model from a historical database (Dahab & Maag, 2019). The possibilities for the application of machine learning techniques for software quality assurance and automatic testing, such are Neural networks (NNs), Support Vector Machine, Clustering, Decision Tree, Grammar Induction, Bayesian Based Method, Random Forest, Generic ML Techniques, have been studied (Chen & Hossain, 2022). Board-based software games, written in object-oriented programming languages, were used for obtaining a regression-based size estimation model for software metrics. The results of the study indicate that forward stepwise multiple linear regression has acceptable accuracy with the value for coefficient of determination (R) of 0.756 (Sabhat et al., 2017).

This study is guided by the idea to use artificial neural networks (ANN) with optimal three-layer back-propagation architecture to identify and find patterns between CK six basic metrics (plus LOC), with the aim to obtain optimal number of input metrics, which will be sufficient to get all the values for basic metrics, with proper accuracy. This approach represents the base for further expansion of metrics sets and model propagation in the cases of object-oriented metrics.

2. Methodology

Study is conducted through following steps:

- determining the set of metrics,
- data normalization,
- determining sets of input, output and hidden layers,
- testing neural networks, and
- choosing the most appropriate set of input variables.

The sample consists from 367 cases of object-oriented metrics taken for programmer's text editor written in Java, jedit 4.2 (Deepti, 2021). These metrics, among the others, include WMC, DIT, NOC, CBO, RFC, LCOM, CA, CE, LOC, CC (with values of max_cc and avg_cc), number of bugs, etc. For the purpose of this study, we have chosen six basic CK metrics, as well as LOC. WMC represents the sum of all complexities for all methods in the given class (Dubey et al., 2012; Ţirban, 2018; Yücalar & Boranda, 2016). DIT measures the length of the path from derived class to its root in the inheritance tree. This value is equal to zero in the case of underived class, while in the case of multiple inheritances DIT represents the distance to the farthest root (Dubey et al., 2012; Molnar et al., 2019; Ţirban, 2018; Yücalar & Boranda, 2016). NOC counts the number of subclasses in the inheritance tree that are directly derived from a given class. It should be used for determining the amount of time that would be spent on class testing (Chidamber & Kemerer, 1994; Dubey et al., 2012; Molnar et al., 2019; Ţirban, 2018; Yücalar & Boranda, 2016;). CBO counts the number of classes from which the observed class depends on (Chidamber & Kemerer, 1994; Dubey et al., 2012; Ţirban, 2018). Number of all methods that might be invoked when calling the methods of an object in the class represents RFC (Chidamber & Kemerer, 1994; Dubey et al., 2012; Molnar et al., 2019; Ţirban, 2018; Yücalar & Boranda, 2016). Compatibility between the methods in the class is expressed with LCOM (Ţirban, 2018; Yücalar & Boranda, 2016). LOC, as a universal software metric, counts the total number of lines (excluding blank lines or comments) in the class, giving the basic information about the project size (Chidamber & Kemerer, 1994; Molnar et al., 2019; Yücalar & Boranda, 2016).

Data about measured values is analyzed using statistical tools and appropriate correlations between them are established. Based on the correlation values WMC, DIT, NOC, CBO, RFC, LCOM, and LOC are separated. WMC and LOC are easily measurable and therefore are taken as input variables, while all other metrics are used as output variables. The values of WMC, DIT, NOC, CBO, RFC, LCOM, and LOC are normalized using robust scaling and used as initial dataset for ANN modeling. These inputs are taken individually or in combination as neurons for the input layer in the ANN. For input layer with 1 neuron WMC metrics or LOC metrics are taken separately, but the results were not satisfactory. Best results are in the case when the input layer has two neurons as their combination. NOC is excluded from the study due the fact that the measured values after robust scaling don't have proper values, so the output layer consists of 4 neurons (DIT, CBO, RFC, and LCOM). Trial-and-error method is used to determine the number of hidden neurons. Proposed NNs are trained with 10 and 20 neurons in hidden layer via back-propagation Levenberg - Marquardt (LM) algorithm, with the aid of nntraintool in Matlab 2018R. LM algorithm proved to be fastest method for training moderate-sized feed forward neural networks. Initial dataset is divided into three parts in relation 257:55:55, respectively for training, validation and testing set, with the aid of function dividerand. Performances for different numbers of hidden neurons are expressed in the form of correlation coefficient (R) and Mean Square Error (MSE).

3. Results and discussion

The best results are in the case of neural network with 2 neurons in input layer, 20 neurons in hidden layer with sigmoid activation function, and 4 neurons in output layer with linear activation function. The values of WMC and LOC metrics represent the inputs' data in the form of 367x2 matrix, while the values of DIT, CBO, RFC, and LCOM metrics represent the target's data in the form of 367x4 matrix. Proposed ANN architecture is presented in Figure 1.

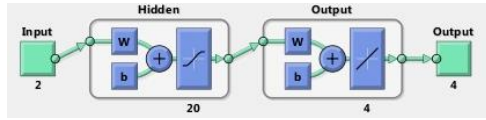

Figure 1. ANN architecture for object-oriented metrics' prediction

Values of R for training is 0.99965, for testing 0.93657, for validation 0.90332, and for all 0.99948 for proposed ANN architecture (Figure 2).

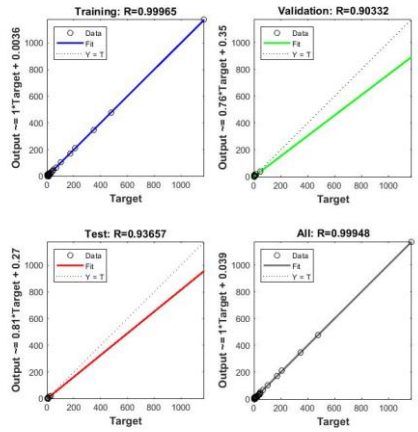

Figure 2. Linear regression between target and predicted values

Based on these values, it can be concluded that the network has been accurately trained, and there is a close relationship between the outputs and the targets. These values for the NN architecture with 10 neurons in hidden layer are 0.99953, 0.93811, 0.84997 and

0.9993. The values for MSE are 1.22988, 2.30203 and 0.629841 for training, testing and validation, while in case of NN architecture with 10 neurons in the hidden layer these values are higher (1.65175, 2.80312, and 1.02848 for training, testing and validation, respectively).

The best validation performance is 2.302 at epoch 11 (Figure 3). The plot shows the L-shaped distribution and follows the horizontal line trends for trained, validation and test data, with negligibly small differences in slope, indicating that there is no overfitting.

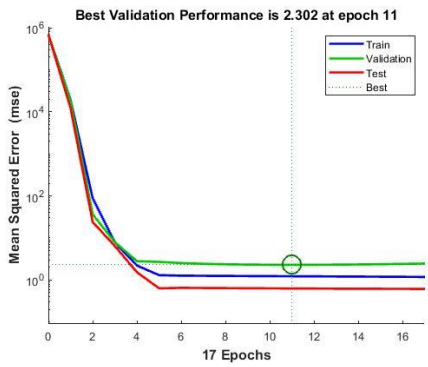

Figure 3. Performance plot

The performance during the learning process in terms of gradient, momentum gain (Mu) and validation checks with respect to number of epochs is shown in Fig. 4. It should be noticed that during the learning process, the gradient from around epoch 7 was near zero, indicating on the fact that there are slight changes in bias and weights which will improve the learning process for the validation. Convergence in ANN training model is achieved after 17 epochs (iterations) with a gradient value of 3.299. Value's changes in Mu, which is used to control weight updating process, are presented for each successive learning epoch. Its value reaches 0.1 at epoch 17, showing that chosen ANN has sufficiently large capacity to predict CK metrics. According to "validation checks" chart (Figure 4), zero values indicate that the learning process was carried out without increasing the values of MSE in the validation phase. Some non-zero values with an upward trend are noticed after 11th epoch, which could be the consequence of the outlier's presence in the sample.

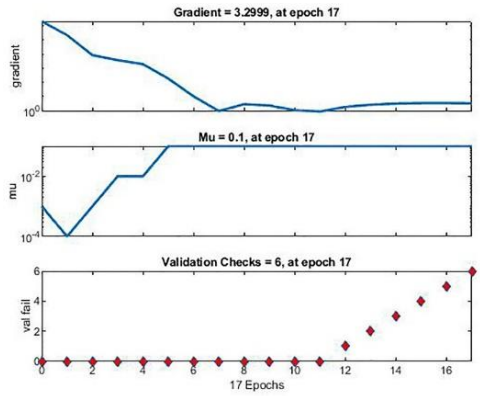

Figure 4. Gradient, momentum, and validation check values during the learning process

4. Conclusion

Proposed ANN model allows quick basic object-oriented metrics' prediction and is valid enough to estimate CK metrics. Taking into account that the values for R indicate a significant and almost perfect adaption of outputs to the targets, some improvements should be made with increasing the number of input/output data, in order to reduce the value of MSE.

Another issue that should be taken into consideration is the fact that it is enough to measure only the values of WMC and LOC metrics, while DIT, CBO, RFC, and LCOM metrics could be predicted. This plays a role in reducing time and effort for obtaining numeric data related to the software projects, which leads to better and faster software quality attributes' assessment, evaluation, and above all, improves the software quality.

References:

Arvanitou, E., Ampatzoglou, A., Chatzigeorgiou, A., & Avgeriou, P. (2016). Software metrics fluctuation: a property for assisting the metric selection process. *Information & Software Technology*, *72*, 110–124. doi: 10.1016/j.infsof.2015.12.010

Badri, M., & Toure, F. (2012). Empirical analysis of object-oriented design metrics for predicting unit testing effort of classes. *Journal of Software Engineering and Applications*, *05(07)*, 513–526. doi:10.4236/jsea.2012.57060

Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, *22*(10), 751–761. doi:10.1109/32.544352

Bos, W. F.A. (2019). Software metrics as indicators for effort of object-oriented software projects. *In 30th Twente Student Conference on IT* (pp 1-9). Febr. 1st, 2019, Enschede, The Netherlands.

Chen, H., & Hossain, M. (2022). Application of machine learning on software quality assurance and testing: a chronological survey. In Eds. Gupta, B., Bandi, A., & M. Hossain. *International Conference on Computers and Their Applications* (pp. 42-52). EPiC Series in Computing, Volume 82.

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, *20(6)*, 476–493. doi:10.1109/32.295895

Dahab, S. A., & Maag, S. (2019). Suggesting Software Measurement Plans with Unsupervised Learning Data Analysis. In Eds. Damiani, E., Spanoudakis, G., & Maciaszek, L. *ENASE 2019: Evaluation of Novel Approaches to Software Engineering* (pp. 189-197). Heraklion, Crete Greece May 4 - 5, 2019. doi:10.5220/0007768101890197

Deepti, A. (2021). Software Defect Prediction Dataset. figshare. Dataset. doi:10.6084/m9.figshare.13536506.v1

Desai, C. G. (2014). Object oriented design metrics, frameworks and quality models. Information Science and Technology, 3(2), 066–070.

Dubey, S. K, Sharma, A., & Rana, A. (2012). Comparison Study and Review on Object-Oriented Metrics. *Global Journal of Computer Science and Technology*, *12(7, Version 1.0)*, 39–48.

Juliano, R. C., Travençolo, B. a. N., & Soares, M. S. (2014). Detection of software anomalies using object-oriented metrics. In *Proceedings of the 16th International Conference on Enterprise Information Systems (ICEIS-2014)* (pp. 241-248). doi:10.5220/0004889102410248

Jureczko, M., & Spinellis, D. (2010). Using object-oriented design metrics to predict software defects. In *Models and Methodology of System Dependability. Proceedings of RELCOMEX 2010: Fifth International Conference on Dependability of Computer Systems DepCoS*, *Monographs of System Dependability* (pp.69-81), Wroclaw, Poland, 2010. Oficyna Wydawnicza Politechniki Wroclawskiej

Kumar, V. P., & Vijayaraghavulu, P. (2021). Software bug prediction using object-oriented metrics. *International Journal of Computers, Electrical and Advanced Communication Engineering*, *10(19)*, 76–82.

Lamba, T., Kavita, & Mishra, A. K. (2017). Optimal metrics selection for software defect Prediction. *International Journal of Data Mining and Emerging Technologies*, *7(2)*, 82-91.

Molnar, A. J., Neamţu, A., & Motogna, S. (2020). Evaluation of Software Product Quality Metrics. In Eds. Damiani, E., Spanoudakis, G., & Maciaszek, L. *Evaluation of Novel Approaches to Software Engineering* (pp. 163-187) Springer, Cham. doi: 10.1007/978-3-030-40223-5_8

Mwangi, W., Joseph, W., & Waweru, S. N. (2014). An effort prediction framework for software code quality measurement based on quantifiable constructs for object oriented design. *International Journal of Computer Trends and Technology (IJCTT)*, *10(1)*, 36-52.

Norbert, N., & András, K. (2021). Review of software quality related ISO standards. *Safety and Security Sciences Review*, *3(2)*, 61–72.

Padhy, N., Satapathy, S. C., & Singh, R. P. (2017). Utility of an object oriented reusability metrics and estimation complexity. *Indian Journal of Science and Technology*, *10(3)*, 1-9. doi:10.17485/ijst/2017/v10i3/107289

Perdomo, W. & Zapata, C. M. (2021). Software quality measures and their relationship with the states of the software system alpha. *Ingeniare. Revista Chilena De Ingeniería*, *29(2)*, 346–363. doi:10.4067/s0718-33052021000200346

Ronchieri, E., Pia, M. G., & Giacomini, F. (2016). Software quality metrics for Geant4: an initial assessment. arXiv (Cornell University). doi:10.48550/arxiv.1608.02008

Sabhat, N., Malik, A. O., & Azam, F. (2017). Utility of CK metrics in predicting size of board-based software games. *Mehran University Research Journal of Engineering and Technology*, *36(4)*, 975–986. doi:10.22581/muet1982.1704.22

Shaik, A., Manda, B., Prakashini, C., Deepthi, K., & Reddy, C. A. (2010). Metrics for object oriented design software systems: a survey. *Journal of Emerging Trends in Engineering and Applied Sciences*, *1(2)*, 190–198. https://journals.co.za/content/sl_jeteas/1/2/EJC136056

Singla, S., & Singh, D. (2014). Classification of Software Metrics and Open Source Tools for Software Development Phase. *IJCSC*, *5(2)*, 103–109.

Thirugnanam, M., & Swathi, J. N. (2010). Quality metrics tool for object oriented programming. *International Journal of Computer Theory and Engineering*, *2*(5), 712–717.

Ţirban, P. (2018). Predicting reliability changes using object oriented metrics, In Ed. Budimac, Z. *Proceedings of the SQAMIA 2018: 7th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications* (pp. 19:1-19:7). Novi Sad, Serbia, 27-30.8.2018.

Vijay, R.R. (2016). *Software metrics tool* (Master thesis, North Dakota State University, Graduate School. Retrieved from https://library.ndsu.edu/ir/bitstream/handle/10365/25820/Software%20Metrics%20Tool.pdf?sequence=1&isAllowed=y

Yücalar, F., & Boranda, E. (2017). Determining the Tested Classes with Software Metrics. *Celal Bayar University Journal of Science*, *13(4)*, 863-871. doi:10.18466/cbayarfbe.330995

Milica Tufegdžić
Academy of Professional Studies Šumadija,
Kragujevac,
Serbia
mtufegdzic@asss.edu.rs

Vladimir Nedić
Academy of Professional Studies Šumadija,
Kragujevac,
Serbia
vnedic@asss.edu.rs

Aleksandar Mišković
Academy of Professional Studies Šumadija,
Kragujevac,
Serbia
amiskovic@asss.edu.rs