Machine learning techniques for predicting software code properties using design metrics

Vira Liubchenko[1,*,†]

[1] Odesa Polytechnic National University, Shevchenka av. 1, 65044 Odesa, Ukraine

Abstract

This paper proposed an information technology to predict code properties based on software design metrics, underscoring the critical interplay between metrics and software code properties. A meticulous case study leveraging data from 39 open-source Java projects demonstrates the efficacy of machine learning methodologies, including random forest and artificial neural networks, in predicting code properties utilizing selected design metrics. The study reveals insights into the correlation between design metrics and lines of code (LOC), suggesting the feasibility of using design metrics for LOC forecasting and, by extension, various software characteristics. The findings emphasize the importance of prioritizing generalizability over specificity to enhance the model's reliability across diverse software engineering contexts. Overall, this paper advances our understanding of the significance of design metrics in forecasting code properties, providing valuable insights into their application within software engineering practices to mitigate risks and enhance software quality. Through these contributions, this research lays a solid foundation for further exploring and utilizing design metrics in software development processes.

Keywords

software quality assurance, predictive modelling, design metrics, performance prediction, machine learning, software engineering, regression analysis, classification techniques, open-source Java projects

1. Introduction

Software quality assurance issues are becoming increasingly important with the spread of software into different spheres of industry and life and with the increasing variety of software types. Therefore, the quest for quality assurance remains paramount. However, achieving these objectives requires more than post hoc debugging and testing; it necessitates proactive measures to predict and preempt potential issues before they manifest. This is where the concept of quantitative prediction emerges as a pivotal tool in the arsenal of software engineers. By leveraging design metrics, architectural insights, and empirical data, engineers could anticipate how a software system will behave under different conditions, enabling them to

IntelITSIS'2024: 5th International Workshop on Intelligent Information Technologies and Systems of Information Security, March 28, 2024, Khmelnytskyi, Ukraine

* Corresponding author.

† These authors contributed equally

lvv@op.edu.ua (V. Liubchenko)

0000-0002-4611-7832 (V. Liubchenko)

optimize its design, allocate resources judiciously, and mitigate potential bottlenecks before deployment.

The importance of performance prediction cannot be overstated in today's software development landscape. In modern IT environments, the traditional approach of reactive troubleshooting proves inadequate, if not impractical. Performance prediction offers a proactive strategy to address these challenges, empowering stakeholders to make informed decisions at every stage of the software development lifecycle.

At the heart of prediction is the premise that specific design metrics are reliable software quality indicators [1]. These metrics encompass various aspects of the software architecture, including its modularity, coupling, cohesion, complexity, and extensibility. By analyzing these metrics, researchers and practitioners seek to discern patterns and correlations that shed light on the system's future performance characteristics.

For instance, studies have shown that high coupling levels between modules tend to increase the propagation of defects and decrease the system's resilience to change [2]. Similarly, as measured by metrics like cyclomatic complexity or code churn, excessive complexity often correlates with higher defect density and lower maintainability. By identifying such patterns, developers can proactively refactor the source code and streamline its structure.

Advances in machine learning and data analytics enable the development of predictive models that leverage historical data to forecast software quality metrics. We can identify underlying trends and accurately predict future outcomes by training these models on repositories of code artefacts, bug reports, and performance logs.

Predicting software performance characteristics based on design metrics holds immense significance. It offers tangible cost reduction, risk mitigation, and stakeholder satisfaction benefits. Software engineers can avoid costly rework, delays, and customer dissatisfaction by identifying potential performance issues early in the software lifecycle. Either limited or absent historical data typically hinders this endeavour. Existing datasets detailing software defects typically lack measurements about the design aspect. Secondly, datasets focusing on design components are often gathered on a per-project basis and lack extensive measurements.

This paper explores methodologies for utilizing indirect metrics to forecast software performance characteristics during the software design phase.

The paper is structured as follows. Section 2 provides an overview of the published research on which our work is based. In section 3, we describe information technology for prediction. Section 4 describes the application of the proposed information technology. Finally, the general conclusions of the work are collected in Section 5.

2. Literature review

Given the paper's focus on software design metrics, we thoroughly analyzed articles about this category of metrics. It's important to highlight that our review exclusively included articles published after 2018.

The paper [3] compared five common design patterns regarding code metrics and time efficiency. It discussed the advantages and disadvantages of each design pattern. It concluded that some metrics might be the same or better with simple solutions, while others might benefit from design patterns. Interestingly, experiments demonstrated the dependencies between design metrics and performance characteristics.

In [4], the authors concluded that object-oriented design metrics can be used as early indicators of software reliability and that the neural network model can effectively model the relationship between metrics and reliability. They suggested that this can help reduce software development and testing costs and effort. The authors used various metrics to measure object-oriented software's design complexity, coupling, cohesion, inheritance, and size.

The authors of [5] described using guide cards based on software metrics to identify four design smells and spectral clustering to group similar smells across different software versions. They analyzed four large Java open-source software projects and demonstrated the possibility of design smell detection based on the software metrics.

The paper [6] investigates the alignment between quality improvement and software design metrics by focusing on eight internal quality attributes and 27 structural metrics. It finds that most design metrics are mapped to the main quality attributes. However, quality attributes, such as encapsulation, abstraction, polymorphism, and design size, are poorly represented by any metrics.

Silva et al. [7] also studied the relationship with software quality attributes but for software architecture metrics. This study confirmed the existence of relationships and allowed the creation of a catalogue of such relationships.

Special consideration should be directed towards the research [8]. The work introduces a novel approach leveraging neural networks to assess software quality characteristics using quality attributes. Internal metrics, which could be estimated using software design, serve as the basis for these quality attributes.

The paper [9] applied software design metrics, such as coupling, cohesion, complexity, and size, to the developer story. It used them as input parameters for supervised machine learning algorithms to predict a class's source code size. The paper conducted a case study based on 30 open-source Java systems and demonstrated the usefulness and effectiveness of metrics for predicting a class's source code size.

The paper [10] investigated the relations among the parameters of object-oriented software metrics using complex network analysis. The authors collected a dataset of object-oriented metrics and their parameters and created a network of parameters based on their co-occurrence in metrics. The authors found that some parameters were more frequently used and more strongly related than others and that the distance between parameters was short regardless of the property they belonged to. The authors concluded that understanding the relationship among software metrics' parameters can help software developers select metrics during the software design phase and that network measurements can be helpful tools for analyzing the relations among software metrics and parameters.

The paper [11] proposed a solution for automating the evaluation of Java project design quality in an educational setting using object-oriented metrics and neural networks. Using manually assigned points as labels, the authors trained a neural network model to predict the design quality score based on the metric values. The model was evaluated on two student projects and homework datasets, and satisfactory results were reported.

The paper [12] proposed a framework to assess the impact of design patterns on software metrics. The results showed no consistent software metrics behaviour between the pattern and non-pattern versions. However, we want to point out that the authors do not deny the metrics' consistent behaviour with the software code's characteristics.

In summarizing the published research, it became evident that all researchers acknowledged the dependency between design metrics and software code properties. The disparity lies in the approaches employed to leverage or model this dependency. Armed with this understanding of the correlation, we can now develop information technology for predicting code properties based on design metrics.

However, we must point out that few studies link design metrics to software characteristics. The main body of publications is devoted to using code characteristics, mainly the number of lines of program code metrics, to predict different software properties.

Lines of code (LOC) is a fundamental software code measure widely used as a proxy for software development effort or as a normalization factor in many other software-related measures [13].

The most developed topic is software defect prediction (SDP). Pradhan et al. [14] used software size in KLOC as an attribute for the SDP model. Effort-aware SDP ranks software modules according to the defect density of software modules, which allows testers to find more defects while reviewing a certain amount of code [15]. This method uses LOC as input data. In [16], authors proposed a deep-learning-based method for predicting potential code defects in software modules. This method used semantic features and LOC simultaneously using the hierarchical LSTM architecture.

However, other applications were described. The experimental results in [17] showed a relationship between testing and LOC. The authors of [18] successfully used LOC for measuring the Maintainability Index. The research [19] used LOC for effort prediction on projects developed with agile methods and a microservice-based architecture.

Based on the analysis performed, we can assume the feasibility of LOC forecasting based on software design metrics. Predicting various software characteristics based on LOC is a well-studied task.

3. Information technology description

Let us generalize the published insights. Information technology aims to predict significant software quality features during the design phase, reducing uncertainty in software projects.

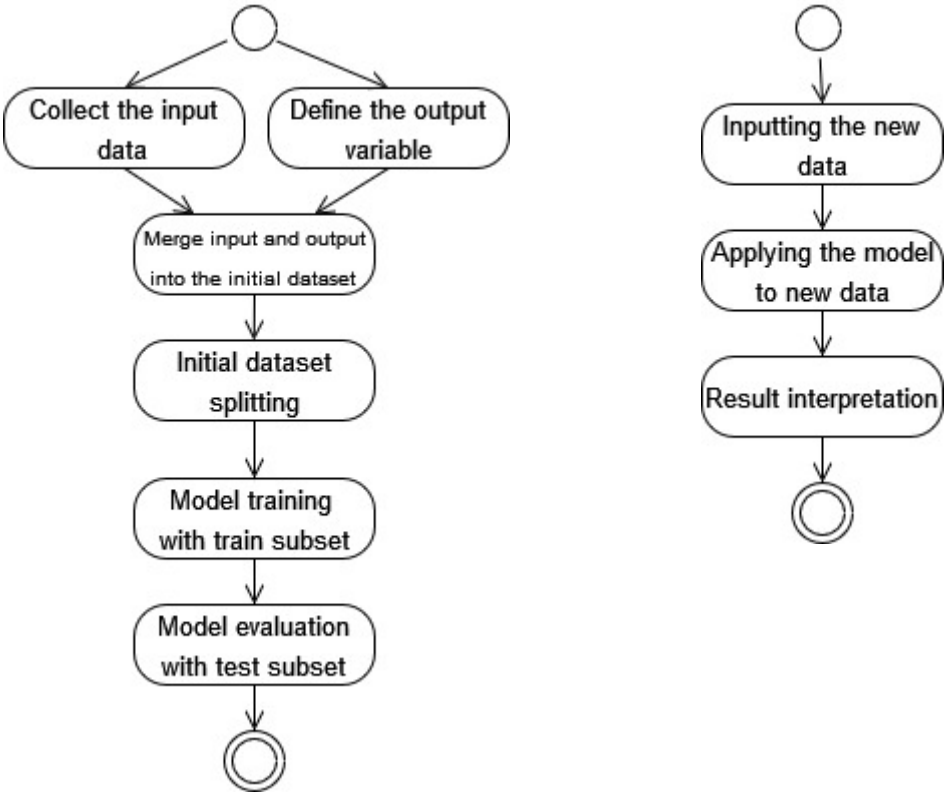
Input data consists of measurements on metrics that describe the software design. Various authors have proposed different sets of metrics, with the most comprehensive list presented in [20]. The results are the quality features that we predict.

A natural constraint on the choice of input and output data is the availability of historical data on the selected metrics. After all, a predictive machine learning model must be trained on historical data.

Information technology comprises two core procedures: model identification and prediction utilizing the model. Both procedures are widely recognized and nearly standard in machine learning. Figure 1 outlines a schematic representation of these procedures.

A sufficient accuracy level of quality feature prediction under specific conditions could always be determined. Depending on this, a particular predicting technique should be chosen.

It's important to note that while the technology itself is significant, the primary focus of our work lies in the results yielded by the case study.



a *b*

Figure 1: Schemas of the model identification (a) and prediction (b) procedures that composed the information technology.

4. Case study

The developed information technology underwent testing using data from 39 open-source Java projects. Data measurements were collected for individual components across 86 metrics within each project. Specific metrics exhibit explicit dependencies, such as the correlation between the number of LOC and kilo lines of program code (KLOC).

Figure 2 illustrates data from three randomly chosen projects plotted in the space defined by the first two principal components. It's important to highlight that the component characteristics of these projects are notably similar, indicating that the datasets from individual projects can be effectively merged.

From the available set of metrics, we used only those related to the software design, namely

- Number of Attributes (NOA),
- Number of Parameters (NOP),
- Number of Children (NOC),
- Coupling Between Objects (CBO),
- Depth Inheritance Tree (DIT),
- Response for a Class (RFC),
- Lack of Cohesion of Methods (LCOM5).

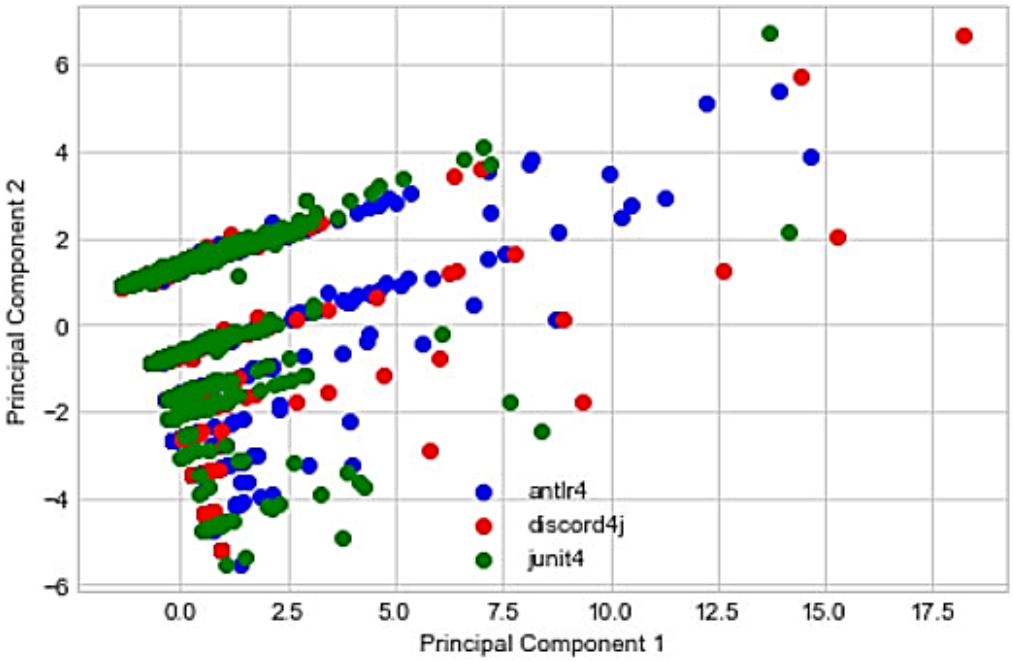


Figure 2: Visualization of projects' data in principal components space.

The datasets provided did not include measurements of metrics directly tied to the quality of a software product, such as the quantity or presence of defects. Nevertheless, it has been established that the number of defects tends to be proportionate to the number of LOCs. This dependency was utilized in our analysis.

Initially, we explored existing linear dependencies by examining the correlation matrix. Consequently, we identified only two significant linear dependencies with LOC: CBO and RFC. Therefore, we opted not to utilize linear regression. Instead, we employed random forest (RF), AdaBoost, and artificial neural network (ANN) to model the regression dependence.

We utilized the Mean Absolute Percentage Error (MAPE) to assess the prediction quality. In our experiment, the AdaBoost method yielded the poorest result, with an error exceeding 100%. The Random Forest (RF) method achieved a MAPE of 40.64%, while the Artificial Neural Network (ANN) allowed for a reduction of MAPE to 30.49%. However, it's important to note that these results alone do not definitively establish neural networks as the superior prediction method.

It's well-known that achieving quality results with neural networks often requires training the model on extensive datasets. Our experiment validated this notion. When the neural network was trained solely on data from one project, the prediction error surpassed 100%, which is comparable to AdaBoost's MAPE. Conversely, the RF and AdaBoost methods exhibited consistent performance across different training dataset sizes—whether trained on data from one, multiple, or all projects.

In summarizing the results obtained, it became apparent that the regression model yielded forecasts of low quality. However, it's worth noting that our primary concern wasn't forecasting LOC itself. Instead, we focused on utilizing LOC to infer quality characteristics. Consequently,

we could establish varying "risk" levels based on module sizes measured by LOC and transition to a classification task. Figure 3 depicts the frequency distribution of LOC values.

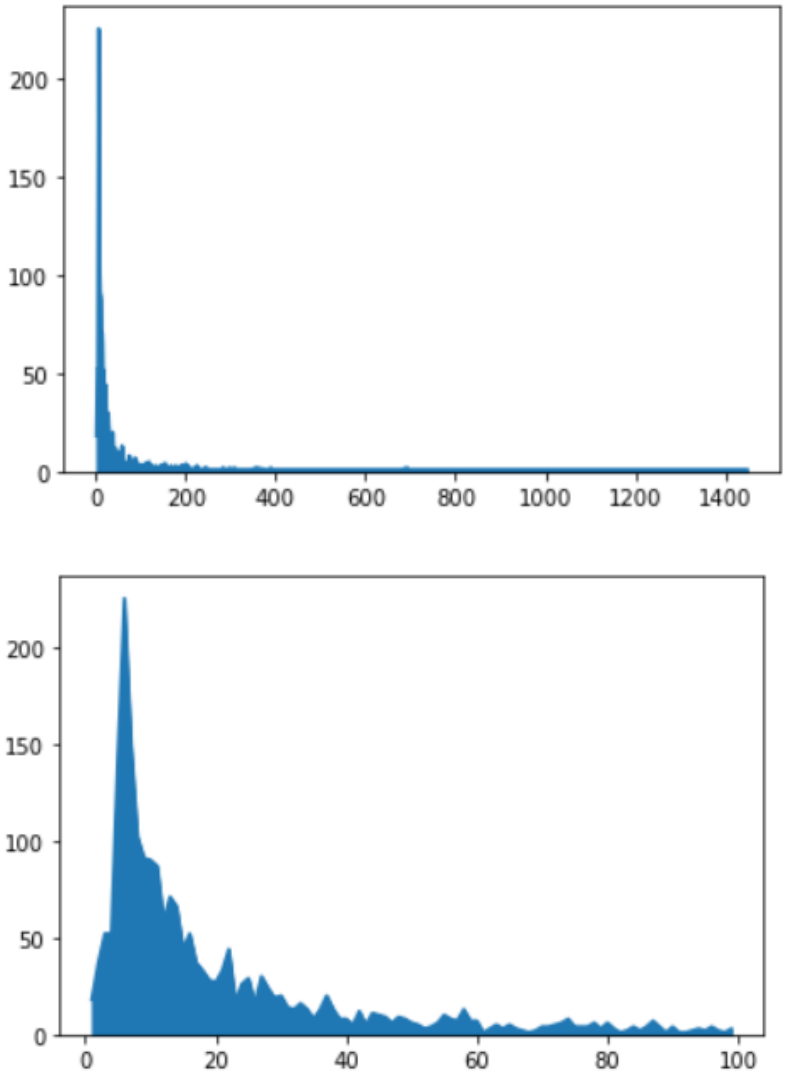


Figure 3: Frequency distribution of LOC values.

Most modules contain up to 75 LOC, suggesting that modules of this size typically exhibit acceptable quality. Conversely, there are a few extensive modules, which likely present challenges. Hence, we propose categorizing modules with LOC>75 as risky.

As it was realized for regression, we used RF, AdaBoost, and ANN methods for the classifier. The results obtained were almost the same (Table 1).

We observe that the ANN demonstrated slightly inferior performance, likely attributed to the inadequacy of the sample size used. For practical applications, RF is preferable due to its consistent performance across datasets of varying sizes and its simplicity compared to the AdaBoost method.

Table 1
Performance measures

Method	Accuracy	Precision	Recall
RF	0.95	0.96	0.98
AdaBoost	0.95	0.96	0.98
ANN	0.95	0.92	0.89

Therefore, reducing the specificity of the outcome criteria enables the development of a model with satisfactory prediction accuracy.

This inference remains valid irrespective of the programming language utilized or the type of project under consideration. By broadening the scope of outcome criteria, the model becomes more adaptable and capable of providing reliable predictions across various contexts.

This flexibility enhances the model's utility and ensures its applicability in diverse software engineering scenarios. Consequently, prioritizing generalizability over specificity enhances the model's effectiveness and reliability in predicting outcomes.

5. Conclusion

The importance of performance prediction cannot be overstated in modern software development environments. Traditional reactive troubleshooting approaches often prove inadequate or impractical. Performance prediction offers a proactive strategy to address these challenges, empowering stakeholders to make informed decisions at every stage of the software development lifecycle.

Advances in machine learning and data analytics have facilitated the development of predictive models that leverage historical data to forecast software quality metrics. We can identify underlying trends and accurately predict future outcomes by training these models on repositories of code artefacts, bug reports, and performance logs.

Predicting software quality based on design metrics offers tangible benefits, including cost reduction, risk mitigation, and stakeholder satisfaction. By identifying potential performance issues early in the software lifecycle, software engineers can avoid costly rework, delays, and customer dissatisfaction.

Reviewing the literature, we find a wealth of research exploring the relationship between design metrics and software quality. Various studies have investigated the efficacy of different metrics and modelling techniques in predicting software reliability, identifying design smells, and assessing the impact of design patterns on software metrics.

Our information technology aims to generalize these insights and provide a framework for predicting significant quality features during the design phase, thereby reducing uncertainty in software projects. By utilizing historical data on design metrics, we can develop predictive models that offer valuable insights into the future performance of software systems.

In our case study, we tested the developed information technology using data from 39 open-source Java projects. By analyzing metrics related to software projects and employing regression and classification techniques, we sought to predict software quality characteristics.

Our results highlight the importance of training predictive models on extensive datasets and the potential limitations of specific modelling techniques.

Ultimately, our findings underscore the value of predictive modelling in software engineering, offering a proactive approach to software quality assurance. Engineers can make informed decisions and optimize software designs by leveraging design metrics and historical data for improved performance and reliability.

References

[1] J. Rashid, T. Mahmood, M. W. Nisar, A Study on Software Metrics and its Impact on Software Quality, in: ArXiv, 2019, abs/1905.12922. URL: https://api.semanticscholar.org/CorpusID:170078652.

[2] P. Skiada, A. Ampatzoglou, E.-M. Arvanitou, A. Chatzigeorgiou, I. Stamelos, Exploring the Relationship between Software Modularity and Technical Debt, in: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Prague, Czech Republic, 2018, pp. 404–407. doi:10.1109/SEAA.2018.00072.

[3] A. Karavokyris, E. Alepis, Software Measures for Common Design Patterns Using Visual Studio Code Metrics, in: 9th International Conference on Information, Intelligence, Systems and Applications (IISA), Zakynthos, Greece, 2018, pp. 1–7. doi:10.1109/IISA.2018.8633694.

[4] C. H. Madhav, K. S. V. Kumar, A method for predicting software reliability using object oriented design metrics, in: International Conference on Intelligent Computing and Control Systems (ICCS), Madurai, India, 2019, pp. 679–682. doi:10.1109/ICCS45141.2019.9065541.

[5] A. Imran, Design Smell Detection and Analysis for Open Source Java Software, in: IEEE International Conference on Software Maintenance and Evolution (ICSME), Cleveland, OH, USA, 2019, pp. 644–648. doi:10.1109/ICSME.2019.00104.

[6] E. A. AlOmar, M. W. Mkaouer, A. Ouni, M. Kessentini, On the Impact of Refactoring on the Relationship between Quality Attributes and Design Metrics, in: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), Porto de Galinhas, Brazil, 2019, pp. 1–11. doi:10.1109/ESEM.2019.8870177.

[7] S. Silva, A. Tuyishime, T. Santilli, P. Pelliccione, L. Iovino, Quality Metrics in Software Architecture, in: 2023 IEEE 20th International Conference on Software Architecture (ICSA), L'Aquila, Italy, 2023, pp. 58–69. doi:10.1109/ICSA56044.2023.00014.

[8] Lebiga, M., Hovorushchenko, T., Kapustian, M. (2022). Neural-Network Model of Software Quality Prediction Based on Quality Attributes. Computer Systems and Information Technologies, (1), 69–74. doi:10.31891/CSIT-2022-1-9.

[9] A. Algarni, K. Magel, Applying Software Design Metrics to Developer Story: A Supervised Machine Learning Analysis, in: IEEE First International Conference on Cognitive Machine Intelligence (CogMI), Los Angeles, CA, USA, 2019, pp. 156–159. doi:10.1109/CogMI48466.2019.00030.

[10] M. M. A. Dabdawb, B. Mahmood, A Network of Object-Oriented Software Metrics' Parameters, in: IEEE International Conference on Communication, Networks and Satellite (COMNETSAT), Purwokerto, Indonesia, 2021, pp. 172–178. doi:10.1109/COMNETSAT53002.2021.9530822.

[11] S. Ćelosmanović, V. Ljubović, JMetricGrader: A software for evaluating student projects using design object-oriented metrics and neural networks, in: 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 2022, pp. 532–537. doi:10.23919/MIPRO55190.2022.9803776.

[12] M. G. Al-Obeidallah, Towards a Framework to Assess the Impact of Design Patterns on Software Metrics, in: International Conference on Multimedia Computing, Networking and Applications (MCNA), Valencia, Spain, 2023, pp. 67–72. doi:10.1109/MCNA59361.2023.10185865.

[13] M. Ochodek, K. Durczak, J. Nawrocki, M. Staron, Mining Task-Specific Lines of Code Counters, in: IEEE Access 11 (2023) 100218–100233. doi:10.1109/ACCESS.2023.3314572.

[14] S. Pradhan, V. Nanniyur, P. K. Vissapragada, On the Defect Prediction for Large Scale Software Systems – From Defect Density to Machine Learning, in: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), Macau, China, 2020, pp. 374–381. doi:10.1109/QRS51102.2020.00056.

[15] J. Rao, X. Yu, C. Zhang, J. Zhou, J. Xiang, Learning to rank software modules for effort-aware defect prediction, in: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C), Hainan, China, 2021, pp. 372–380. doi:10.1109/QRS-C55045.2021.00062.

[16] H. Wang, W. Zhuang, X. Zhang, Software Defect Prediction Based on Gated Hierarchical LSTMs, in: IEEE Transactions on Reliability 70(2) (2021) pp. 711–727. doi:10.1109/TR.2020.3047396.

[17] S. Ahmed, L. Sadath, J. Nagaria, Software Testing and Lines of Codes-A Study on Software Engineering Design Patterns, in: 2019 International Conference on Automation, Computational and Technology Management (ICACTM), London, UK, (2019) pp. 389–394. doi:10.1109/ICACTM.2019.8776688.

[18] G. H. Kencana, A. Saleh, H. A. Darwito, R. R. Rachmadi, E. M. Sari, Comparison of Maintainability Index Measurement from Microsoft CodeLens and Line of Code, in: 2020 7th International Conference on Electrical Engineering, Computer Sciences and Informatics (EECSI), Yogyakarta, Indonesia (2020) pp. 235–239. doi:10.23919/EECSI50503.2020.9251901.

[19] H. Ünlü, T. Hacaloglu, F. Büber, K. Berrak, O. Leblebici, O. Demirörs, Utilization of Three Software Size Measures for Effort Estimation in Agile World: A Case Study, in: 2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Gran Canaria, Spain (2022) pp. 239–246. doi: 10.1109/SEAA56994.2022.00045.

[20] E. Y. Hernandez-Gonzalez, A. J. Sanchez-Garcia, M. K. Cortes-Verdin, J. C. Perez-Arriaga, Quality Metrics in Software Design: A Systematic Review, in: 7th International Conference in Software Engineering Research and Innovation (CONISOFT), Mexico City, Mexico, 2019, pp. 80–86. doi: 10.1109/CONISOFT.2019.00021.