Grenze International Journal of Engineering and Technology, June Issue

A thorough Examination of a Software Effort Estimating Model based on Deep Learning

Anup Narendra Kadu[1] and Nisha Wandile Kimmatkar[2]
[1-2]JSPM's Rajashri Shahu College of Engineering, Tathawade, Pune-33
Email: anupkadu1306@gmail.com, nishakimmatkar@gmail.com

***Abstract*—Due to immense growth in digital technology and high-end intelligence algorithms, the software industry has grown immensely in a multi-fold manner over the years. This immense growth involves thousands of software engineers and developers to contribute their skills in coding in the process of product development, delivery, deployment, and services. More often, due to the wrong estimation of the software efforts, the development companies may have deep holes in their pockets. Traditional techniques like the COCOMO model and others are working in a sequential manner to assess the software effort estimation; this was good in an earlier way of software development. Due to the evolution of cloud computing, the software development process has become deeply distributed. Hence, an intelligent way is required to assess the software effort estimation process, which involves the deep learning model and more. So, this survey paper focuses primarily on evaluating the earlier works in software effort estimation and trying to find the major gaps to propose a better model to achieve good accuracy. This research paper proposes a strategy for finding the software effort estimation by using support vector machine model with a hybrid model powered by an artificial neural network (ANN) and long-short-term memory (LSTM) to catalyze the decision tree model to produce good accuracy on software effort estimation.**

***Index Terms*— Artificial Neural network (ANN), Long-short term memory (LSTM), Support Vector Machine(SVM), Software effort estimation, COCOMO model.**

I. INTRODUCTION

Software project development involves a variety of tasks that must be completed within a given time frame and budget, from requirement collecting to testing and maintenance. Logical and analytical work is the foundation of software engineering. The fast rate of change in customer needs and the quick improvement of technology make software development more challenging than other types of engineering projects. As a result, achieving certain goals while meeting a variety of restrictions becomes difficult for good software project management. The scenario where hardware is cheaper than programming was brought about by the rapid trend in hardware innovation. In order to achieve noticeable performance gains, software optimization is necessary regardless of hardware speed. This means that developers must create flexible software within limited resources and timeframes in order to close the gap between software and hardware development. One such area that is developing quickly is the realm of technology. Trends in software development such as virtual reality, blockchain, mobile computing, artificial intelligence, autonomous cars, and cyber security have already emerged in the modern period. Furthermore, the development of network technology has led to a tendency toward

Grenze ID: 01.GIJET.10.2.538_1

managerial simplification and programmability, which fosters innovation. In large and dispersed enterprises, planning and estimating tasks can be very difficult.

For developers as well as project managers, it can be difficult to accurately estimate requirements and gauge the software's level of sophistication during the early phases of development. Processes to calculate project cost, effort in manhours, and time to completion are all included in software cost and effort estimation. Underestimating the effort required for a project can lead to overspending, underdeveloped or limited functionality, inferior quality, and an inability to finish it on schedule [5]. Because too many resources are committed to the project, overestimating will lead to resource waste. As a result, accurate assessment of time, cost, and effort is not only regarded as essential to a project's success but also useful for making business decisions. Accurately understanding requirements is critical to the software project's success. These requirements include (i) impacts on the system design; (ii) determining performance bounds due to hardware advancement; (iii) laying out assumptions for the new environment; (iv) laying out dynamic changes to customer requirements; and (iv) looking for the best trade-off.

Numerous methods for estimating costs and efforts have been proposed by previous research. Even after all the effort, accurate effort estimate remains a challenge. Two general categories can be used to categorize current software effort estimation research efforts: (i) non-algorithmic and (ii) algorithmic. The following methods are based on non-algorithmic models: expert judgment, fuzzy logic, artificial neural networks (ANNs), machine learning, analogy-based estimate, and case-based reasoning (CBR). Over the past few decades, the software development process has advanced.

Because their sources come from a range of organizational projects and because there are a lot of missing variables, the majority of the datasets that are now available are therefore heterogeneous. Classifying high-dimensional, multi-objective data using neural networks (NNs) is a difficult undertaking.

Mathematical and statistical models that make use of project, product, and process-related factors are used in algorithmic models to give effort estimation. Several examples are Putnam's Function Point Analysis (FPA), Constructive Cost Model (COCOMO), Software Lifecycle Model (SLIM), Software Evaluation and Estimation of Resources - Software Estimation Model (SEERSEM). The estimation of effort pertaining to the creation of new software is grounded in historical data from prior software projects. One of the most popular algorithmic methods for software effort estimation based on regression analysis is COCOMO. Based on variables like size, staff experience, and software type, COCOMO separates projects into three groups. When allocating starting values to the parameters, COCOMO takes the project category into account.

As it happens, data on a variety of projects is included in the software project datasets that are now accessible. It is challenging to have a single, reasonable, and workable parametric model for this reason. Thus, fine-tuning the parameters is necessary to ensure estimation accuracy.

The work that has already been done on software effort estimation to improve effort estimate and address the shortcomings of COCOMO models uses a variety of techniques to adjust the coefficients. Recent literature has shown an increasing tendency toward the optimization of software effort estimation and the adjustment of COCOMO coefficients through the use of meta-heuristic techniques. Because of their special characteristics, which include a vast searching space and a random selection technique, these meta-heuristic algorithms fared well while handling the optimization problem in several sectors of interest.

A review of the literature on software efforts reveals an increase in the amount of research being done on machine learning-based software development effort estimation. The scientific community is currently exploring Deep Learning (DL) as a potential means of enhancing cost estimation. Deep Neural Networks (DNNs) are a better option for software cost prediction because they can reflect intricate correlations between effort and cost factors.

[1] N. The goal of Rankovic et al. was to increase the precision of software assessment techniques. In order to estimate software effort and costs, this paper compares the effects of two artificial neural network designs that are presented as nonparametric models to classic parametric models like COCOMO2000. The COCOMO81, COCOMO2000, and Kemerer datasets—actual, publicly available datasets—were utilized for comparison. There is a link in the decision-making process between the planning criteria and performance measurements. There could be a weight component to this association. The relative significance of the scheduling criterion is shown by these weights. The operational policy is outlined by the factors and the weights assigned to them. In order to ascertain the relationship between the global performance metrics in relation to the operational policies and the work orders that need to be scheduled, a backpropagation neural network was selected for this experimental investigation. Most of the time, a Neural Network methodology was selected in response to analytical and simulation methods that are neither practical nor economical. The usage of various ANN architectures, as well as the training, testing, and validation of such structures on additional datasets, the cost-effect function's value, and

other parameters, will be the main topics of future research. System will take into consideration the use of orthogonal vector plans for error minimization for numerous input values. The application of the Taguchi technique in this analysis offers a scientific advancement for evaluating the backpropagation neural network's accuracy after receiving a set of input data and for increasing the network's sensitivity when it comes into contact with various levels. In order to create the matrix experiments that offer an extra trustworthy assessment of the planning factors, it also entails the use of orthogonal arrays.

[2] According to E. Cibir et al., estimating the labor required for software testing is essential to its successful execution. Due to both internal and external considerations, software test effort estimation is particularly difficult and complex in defense projects. The purpose of this study is to look into the connections between industry-standard software test metrics and software testing initiatives. A fresh collection of software testing metrics, not previously employed in any suggested software test effort estimation methodologies, is proposed as a way for estimating the testing effort. This study examined 15 finished software projects from a military industry company that was certified by CMMI Level-3. With Pred (0.25) and Pred (0.30) values equal to 0.867, the empirical study's findings demonstrate that the suggested strategy using the provided metrics offers a prediction quality that is acceptable. The system produced believable findings and showed that the newly suggested metrics may be utilized for software test effort estimation without risk.

[3] Based on the 49 primary papers that were chosen from IEEE Xplore, Science Direct, ACM Digital Library, and Springer Link, M. Ali et al.'s comprehensive literature review offers insightful information about the critical role that feature selection techniques play in providing software defect predictions. It tackles important research questions and draws attention to prevailing patterns including the popularity of hybrid FS techniques and filters. It also looks into how specific classifiers like NB, SVM, and DT are used. The use of ensemble classifiers like RF and bagging is also covered. The review also explores the use of several measures for performance evaluation. The report also highlights the tools of choice, which include Python, MATLAB, and WEKA. These results provide a thorough grasp of the current trends in feature selection techniques and set the stage for further study to raise the precision and effectiveness of models used to predict software defects. To improve the efficacy of software defect prediction models, future research should examine the effects of different feature selection techniques on ensemble model performance.

[4] Three ML-based software modules were built as part of an experiment by A. O. Sousa et al. with the aim of assessing the applicability and efficacy of ML-based methodologies to estimate the effort and duration of particular tasks or issues in software projects. In SCRAIM, Project Control, and JIRA, three project management applications, they offer estimates on issue effort and duration. To improve the relevance of the comparisons and analysis carried out, the experimental setting was kept as uniform as feasible despite the fact that the datasets employed were diverse. In an effort to identify algorithms that consistently performed well, the same collection of algorithms was also evaluated across the three software components. System discovered that ensemble-based algorithms, including Random Forest, Gradient Boosted Trees, Extra Trees Regressor, and XG Boost, consistently produced the best results across all datasets after examining the test results. They were consistently among the top four algorithms for the majority of the evaluation criteria, even if the order changed based on the dataset and the assessment metric. System intends to investigate further text preparation methods in the future, specifically pre-trained language models like BERT.

[5] MANSOOR AHMED et al. explains how estimating the software development effort is a critical task in software management. Owing to the significance of software effort estimation, numerous approaches and models have been put forth; yet, no single optimum approach or model for software work estimation has been documented. The most widely used estimating techniques are divided into two categories: group techniques and analogy-based techniques. Nevertheless, both have several drawbacks that could influence the predicted outcomes, such as historical data, a lack of expertise, and group bias. At the moment, the employment of cutting-edge technology to enhance the estimation outcomes is appreciated by the research community. In light of the new circumstances, this study presents the Blockchain-Based Software Effort Estimation (BBSEE) approach, which aims to improve estimation outcomes by addressing the drawbacks of group and analogy-based estimation. System specialists from 52 organizations participated in two case studies that replicated the planned BBSEE. According to the first case study's findings, the main obstacles preventing organizations from doing estimating tasks more successfully are a lack of expertise, a lack of historical data, and group bias. The second case study's findings lend credence to the theory that the project manager may use BBSEE to help him or her overcome issues including a lack of experience, member bias, a lack of historical data, and estimation based on analogies.

The structure of this literature review study is as follows: In the form of a review of the literature, Section 2 examines earlier research; Section 3 delineates the recommended approach; and Section 4 makes recommendations for more research.

II. Literature Survey

[6] Software effort estimation is necessary for software development projects related to industrial software systems and digital transformation initiatives, according to A. Jadhav et al. The process of integrating digital technologies into different areas of a company or organization to improve processes, customer experiences, operations, and overall performance is known as digital transformation. This work aims to provide a reliable and efficient machine learning-based effort prediction model. SMAPE, MRE, MASE, NSE, and COD values—evaluation metrics—show that the proposed Omni-Ensemble Selection (OES) performs better than individual machine learning models over the Finnish and Maxwell datasets.

[7] K. Zhang et al. introduced a unique method using a deep learning-based function point type classification model to increase the effectiveness of sentence-level function point analysis. Using BiLSTM-CRF, the method extracts function point features from phrases and categorizes them into various kinds in accordance with the FPA standard. A quantitative assessment of the efficiency increase and validation of the suggested method were carried out through an industry partner case.

[8] To improve the performances of machine learning models in fault-proneness prediction, KHOA PHUNG et al. have examined the possible benefits and drawbacks of the ESM values, which were first introduced in as a new set of software metrics called Error-type software metrics. Additionally, System has put forth an approach that makes use of machine learning and Stream X-Machine to model, retrieve, and assess the ESM values. Using software measurements acquired from the Bug Hunter Dataset, System ran experiments on the JUnit project using four machine learning models: Decision Tree, Logistic Regression, Multilayer Perceptron, and Naïve Bayes.

[9] According to H.D. P. De Carvalho et al., software engineers use the project management method to help managers maintain control over their projects. Establishing a precise and trustworthy estimate of the work needed to finish the project is one of the fundamental steps in software engineering. The goals of this paper are to: i) use correlation analysis to determine which variables affect estimation; and ii) apply the Extreme Learning Machine (ELM) model for effort estimation and compare its results with those of previous studies. Therefore, the method with the highest accuracy for effort prediction was studied. Based on statistical tests and absolute mean residue (MAR) criteria predictive precision, the models were compared to one another. The study's primary conclusions were that: i) there are significant effort estimation factors; and ii) the ELM model outperforms other models in the literature in terms of software design effort estimation. In this sense, the accuracy of the time estimates and project costs can be increased by utilizing machine learning techniques in the effort estimating process.

[10] S. S. Ali along with others. Using the ensemble technique to increase the accuracy of performance estimations for software development effort is the main objective of this study. System offered a cutting-edge Heterogeneous Ensemble model for software work estimation. A heterogeneous ensemble model that integrates use case points (UCP), expert judgement (EJ), and artificial neural networks (ANN) is presented and applied on a benchmark dataset in order to improve the accuracy and efficiency of software development projects. Industrial case studies and ISBSG. The system has employed several variations when utilizing the dataset, not only to prevent bias in the results but also to see variations in the outcomes. Observing how machine learning algorithms respond to various experimental configurations was an intriguing observation. In the course of testing the models, System discovered that ensemble models outperformed standalone models on the benchmark datasets.

[11] A novel transformation and feature selection strategy is put forth by Y.Z. Bala et al. to enhance cross-project defect prediction performance. Reducing the disparity in distribution between the source and target projects was the primary goal of the transformation, while reducing high dimensionality and irrelevant features was the primary goal of the feature selection process. During the transformation phase, the source project's mean distributional characteristic was used in step one; the source's mode distributional characteristic was used in step two; and, in the final step, the arithmetic mean of the two was calculated and taken into consideration as the transformed source. Each modified source feature's distance was calculated during the feature selection phase using three times the standard deviation. Large-distance features are chosen. With the highest mean F-score of any CPDP approach now in use, 0.80, the model's approach successfully increased CPDP performance.

[12] The estimated effort by N.M. Alsheikh et al. has a big influence on cost estimation. Many academics in the field of software engineering have tried to present techniques and models for accurately evaluating work. System's ability to accurately estimate effort helps us to complete projects on time and within budget. The Grey Wolf Optimization is used in this work to improve the basic-COCOMO coefficient values. Additionally, the outcomes were contrasted with those of the PSO, Genetic, and Firefly algorithms. Moreover, ZOA, PDO, WSO, and MFO are the other four methods used to assess how well the GWO performs in determining the ideal effort estimation value. Additionally, a comparison between the results and the GWO results has been done. The optimized models are assessed using VAF, MSE, MAE, MMRE, RMSE, and R 2 evaluation measures.

[13] According to M. Daud et al., estimating the size of software early on in the development process might be difficult due to the lack of information available. The purpose of this study was to solve this difficulty by quantifying the effect of new information introduced during the transition from the analysis to the design phases on early software size estimation. For this reason, a brand-new metrics class known as analysis-todesign adjustment factors (ADAFs) was created.

Four distinct class diagram metrics—NOC, NOA, NOM, and NOR—that are frequently employed in various class diagram-based software size estimation models had their ADAFs computed. 84 projects were used to conceptually and experimentally validate these ADAFs. Furthermore, the applicability of ADAFs to actual industry initiatives demonstrates their practical value. The accuracy of current early software size estimation models was evaluated before and after the application of ADAFs by System in order to evaluate the usefulness of these ADAFs in early software size estimation.

[14] According to M.S. Khan et al., the most important task for the success of software engineering projects' total solution delivery is effort estimation. The work makes two key contributions to the literature on software effort estimation in this regard. This study first looks at using meta-heuristic methods to create a parametric model for software effort estimation that makes sense and is acceptable. Second, a Deep Neural Network (DNN) model for software effort estimation based on meta-heuristic algorithms is presented in this paper in order to explore the advantages of using nature-inspired meta-heuristic algorithms in improving Deep Learning (DL) architectures for software effort estimation. This study applies the Straw Berry (SB) and Grey Wolf Optimizer (GWO) meta-heuristic algorithms to obtain a reasonable and logical parametric model for software effort estimation. The performances of these two algorithms are verified using a set of nine benchmark functions with large dimensions. Five alternative meta-heuristic techniques for software effort estimation that have been utilized in the literature are compared with the results of the GWO and SB algorithms. The outcomes of the experiment demonstrated the GWO's overall superiority in terms of estimating accuracy. In comparison to previous work on employing DNN for software effort estimation, the suggested DNN model (GWDNNSB) provided improved results by selecting the learning rate and initial weights using meta-heuristic techniques.

[15] According to G.Molan et al., there is a growing disparity between the resources that software developers have access to and the requirements for software development. This necessitates adjustments to the process and structure of software development. In order to calculate the ultimate worth of the software product, this study presented a quantitative framework for software development management that assesses the relative relevance and risk of functionality retention or abundance. A software product's ultimate worth is deduced from its specifications and features, which are depicted as a computational graph known as the software product graph.

[16] According to H. L. T. K. Nhung et al., the OCF approach should be modified in order to achieve more accurate estimations. The idea behind the proposed ExOCF technique is to simplify the original principles of the UCP that the OCF method is having while addressing the issue of human error influencing UCM analysis through the use of a standard estimation procedure. In particular, the system built regression models and reduced mistakes in the integration or recursion process by using MLR models on past project data points. The suggested approach reduces prediction error and enhances the OCF methods' capacity to estimate software size.

[17] C. H. Rashid along with others. have used ML and non-ML techniques over the last fifteen years to conduct a systematic literature review on software cost and effort estimation. 52 studies in all were found and examined by the study from five well-known digital libraries. Even though there have been a lot of SLRs published, there has been a lot of work in the field of software effort estimating over the last 15 years that needs to be examined. Additionally, there are variations in the methodologies and procedures used by various SLRs. In order to suggest a future course of action, this SLR has been undertaken to investigate the current research trends in software cost and effort estimate. It aims to identify the most popular estimating methodologies, datasets, and application domains.

[18] V. Tawosi et al. have expanded and replicated an earlier study, referred to as CoGEE, which represents the state-of-the-art in multi-objective software effort estimating at the moment. The first author, who was not involved in the initial study, conducted an independent implementation of the system to investigate the original

research objectives and included a more robust baseline (LP4EE). System enhances the original study's internal and external validity through its replication.
According to V. Van Hai et al. [19], project managers face a major challenge when it comes to accurately estimating software effort during the software development process. For stakeholders, inaccurate forecasting might result in either an overestimation or an underestimation of software work. One of the most important techniques for estimating software effort is the Function Point Analysis (FPA) method developed by the International Function Point Users Group.
[20] L. According to Rojas et al., requirements prioritizing is a process used to identify the minimum set of needs that must be included in a software release. Many prioritization techniques exist to help organize this effort, but there are still issues that need to be worked out. The expectations and objectives of stakeholders are not always reached by the requirements prioritizing process using current methodologies. This is due to the fact that the majority of prioritization techniques only take into account quantitative data, which makes it challenging to officially capture the interests and viewpoints of stakeholders that are better expressed in qualitative terms. Similarly, approaches that incorporate qualitative data only take into account factors linked to benefit estimation, or the good parts of the project, ignoring the bad or expensive parts. As a result, a limited understanding of restrictions influences the prioritization process. Additionally, many approaches fall short in gathering and integrating expert knowledge that decision-makers can include into the process. In this study, System presents an innovative approach to software requirements prioritizing that takes cost and benefit restrictions into account and makes it easier to incorporate experts' qualitative assessments at the beginning of the process. The approach is described in detail, and a case study that describes an actual application scenario is included. Based on the case study's findings, recommendations and suggestions for using the method are put forth.

III. PROPOSED MODEL

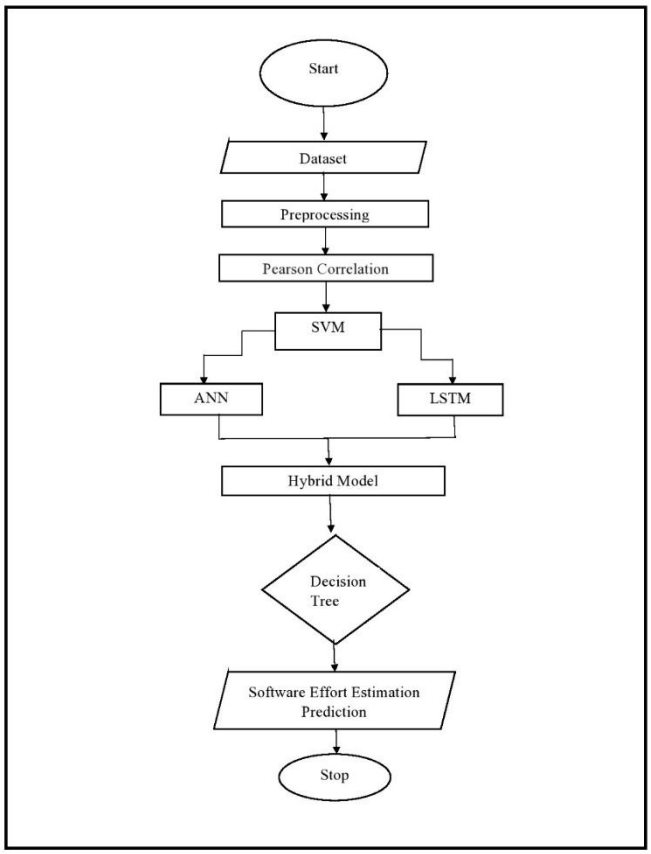

Figure 1: Proposed model

The procedure shown in the flowchart above is achieved by attentively examining the existing approaches in software effort estimation. The process starts at the outset when the system receives an input dataset that contains attributes related to software effort estimation. The input dataset is pre-processed to extract the relevant data attributes. Using the Pearson correlation, the extracted data attributes are used to estimate the correlation and extract the quality attribute. The support vector machine is used to classify the quality characteristics, and the classified attributes are subsequently input into deep learning models such as long-short-term memory models and artificial neural networks (ANN). To generate the best software effort estimation results, a hybrid model that uses the decision tree technique is trained using the provided data.

IV. Conclusion and Future Scope

In software engineering projects, effort estimating is the most important task for the total solution delivery process to succeed. The work makes two key contributions to the literature on software effort estimation in this regard. This study first looks at using meta-heuristic methods to create a parametric model for software effort estimation that makes sense and is acceptable. The second objective is to examine the advantages of using meta-heuristic methods inspired by nature for optimizing Deep Learning (DL) architectures in software effort estimation. Thus, the main objective of this survey study is to assess previous software effort estimating efforts and identify any significant gaps so that a more accurate model can be proposed. In order to generate good accuracy in software effort estimation, this research paper suggests a method for estimating software effort using a hybrid model that combines an artificial neural network (ANN) and long-short-term memory (LSTM) to catalyze the decision tree model.

The findings of the suggested model will also be available in a later version of this work. In the future, this procedure may be used in real-time effort evaluation based on the locations of the enterprises by taking the socioeconomic level of the location into consideration.

References

[1] N. Rankovic, D. Rankovic, M. Ivanovic and L. Lazic, "A New Approach to Software Effort Estimation Using Different Artificial Neural Network Architectures and Taguchi Orthogonal Arrays," in IEEE Access, vol. 9, pp. 26926-26936, 2021, doi: 10.1109/ACCESS.2021.3057807

[2] E. Cibir and T. E. Ayyildiz, "An Empirical Study on Software Test Effort Estimation for Defense Projects," in IEEE Access, vol. 10, pp. 48082-48087, 2022, doi: 10.1109/ACCESS.2022.3172326.

[3] M. Ali et al., "Analysis of Feature Selection Methods in Software Defect Prediction Models," in IEEE Access, vol. 11, pp. 145954-145974, 2023, doi: 10.1109/ACCESS.2023.3343249.

[4] A. O. Sousa et al., "Applying Machine Learning to Estimate the Effort and Duration of Individual Tasks in Software Projects," in IEEE Access, vol. 11, pp. 89933-89946, 2023, doi: 10.1109/ACCESS.2023.3307310.

[5] MANSOOR AHMED, NAEEM IQBAL, FARAZ HUSSAIN, MURAD-ALI KHAN , MARKUS HELFERT , IMRAN , AND JUNGSUK KIM.” Blockchain-Based Software Effort Estimation: An Empirical Study ”, Received 12 September 2022, accepted 29 September 2022, date of publication 25 October 2022, date of current version 21 November 2022.

[6] A. Jadhav, S. K. Shandilya, I. Izonin and M. Gregus, "Effective Software Effort Estimation Leveraging Machine Learning for Digital Transformation," in IEEE Access, vol. 11, pp. 83523-83536, 2023, doi: 10.1109/ACCESS.2023.3293432.

[7] K. Zhang, X. Wang, J. Ren and C. Liu, "Efficiency Improvement of Function Point-Based Software Size Estimation With Deep Learning Model," in IEEE Access, vol. 9, pp. 107124-107136, 2021, doi: 10.1109/ACCESS.2020.2998581.

[8] KHOA PHUNG , EMMANUEL OGUNSHILE , AND MEHMET AYDIN , “A Novel Set of Software Metrics for Software Fault Prediction”, Received 23 February 2023, accepted 23 March 2023, date of publication 27 March 2023, date of current version 30 March 2023.

[9] H. D. P. De Carvalho, R. Fagundes and W. Santos, "Extreme Learning Machine Applied to Software Development Effort Estimation," in IEEE Access, vol. 9, pp. 92676-92687, 2021, doi: 10.1109/ACCESS.2021.3091313.

[10] S. S. Ali, J. Ren, K. Zhang, J. Wu and C. Liu, "Heterogeneous Ensemble Model to Optimize Software Effort Estimation Accuracy," in IEEE Access, vol. 11, pp. 27759-27792, 2023, doi: 10.1109/ACCESS.2023.3256533.

[11] Y. Z. Bala, P. Abdul Samat, K. Y. Sharif and N. Manshor, "Improving Cross-Project Software Defect Prediction Method Through Transformation and Feature Selection Approach," in IEEE Access, vol. 11, pp. 2318-2326, 2023, doi: 10.1109/ACCESS.2022.3231456.

[12] N. M. Alsheikh and N. M. Munassar, "Improving Software Effort Estimation Models Using Grey Wolf Optimization Algorithm," in IEEE Access, vol. 11, pp. 143549-143579, 2023, doi: 10.1109/ACCESS.2023.3340140.

[13] M. Daud and A. A. Malik, "Improving the Accuracy of Early Software Size Estimation Using Analysis-to-Design Adjustment Factors (ADAFs)," in IEEE Access, vol. 9, pp. 81986-81999, 2021, doi: 10.1109/ACCESS.2021.3085752.

[14] M. S. Khan, F. Jabeen, S. Ghouzali, Z. Rehman, S. Naz and W. Abdul, "Metaheuristic Algorithms in Optimizing Deep Neural Network Model for Software Effort Estimation," in IEEE Access, vol. 9, pp. 60309-60327, 2021, doi: 10.1109/ACCESS.2021.3072380.
[15] G. Molan, G. Dolinar, J. Bojkovski, R. Prodan, A. Borghesi and M. Molan, "Model for Quantitative Estimation of Functionality Influence on the Final Value of a Software Product," in IEEE Access, vol. 11, pp. 115599-115616, 2023, doi: 10.1109/ACCESS.2023.3325338.
[16] H. L. T. K. Nhung, V. Van Hai, R. Silhavy, Z. Prokopova and P. Silhavy, "Parametric Software Effort Estimation Based on Optimizing Correction Factors and Multiple Linear Regression," in IEEE Access, vol. 10, pp. 2963-2986, 2022, doi: 10.1109/ACCESS.2021.3139183.
[17] C. H. Rashid et al., "Software Cost and Effort Estimation: Current Approaches and Future Trends," in IEEE Access, vol. 11, pp. 99268-99288, 2023, doi: 10.1109/ACCESS.2023.3312716.
[18] V. Tawosi, F. Sarro, A. Petrozziello and M. Harman, "Multi-Objective Software Effort Estimation: A Replication Study," in IEEE Transactions on Software Engineering, vol. 48, no. 8, pp. 3185-3205, 1 Aug. 2022, doi: 10.1109/TSE.2021.3083360.
[19] V. Van Hai, H. L. T. K. Nhung, Z. Prokopova, R. Silhavy and P. Silhavy, "Toward Improving the Efficiency of Software Development Effort Estimation via Clustering Analysis," in IEEE Access, vol. 10, pp. 83249-83264, 2022, doi: 10.1109/ACCESS.2022.3185393.
[20] L. Rojas, C. Olivares-Rodríguez, C. Alvarez and P. G. Campos, "OurRank: A Software Requirements Prioritization Method Based on Qualitative Assessment and Cost-Benefit Prediction," in IEEE Access, vol. 10, pp. 131772-131787, 2022, doi: 10.1109/ACCESS.2022.3230152.