BULLETIN OF INFORMATION TECHNOLOGY (BIT)
Vol 6, No 2, Juni 2025, Hal. 126 - 135
ISSN 2722-0524 (media online)
DOI 10.47065/bit.v5i2.2027
https://journal.fkpt.org/index.php/BIT

Evaluation Of COCOMO Model Accuracy In Software Effort Estimation

Umar Jeklin*, Muhammad Ibnu Saad, Hanifah Ekawati
Widya Cipta Dharma, Informatics, STMIK Widya Cipta Dharma, Samarinda, Indonesia
Email: *2143021@wicida.ac.id, saad@wicida.ac.id, hanifah@wicida.ac.id
Correspondence Author Email: umarjekliin@gmail.com

Abstract- Accurate effort estimation underpins on-time,on-budget software delivery. This study empirically assesses the baseline Constructive cost Model (COCOMO) by applying standard organic-mode parameters (a = 2.4, b = 1.05) to the COCOMONASA dataset, which contains 63 NASA projects ranging from 2 KLOC to 100 KLOC. Model ourputs are benchmarked against recorded person-month effort using Mean Absolute Error (MAE), Mean Magnitude of Relative Error (MMRE), and Predcitions at 25 percent error (PRED 0.25). Results show MAE values 295-661 person-months and an MMRE near 1.0, indicating average relative error of ~100 percent. PRED (0.25) equals 0.0, meaning no project is estimated within the industry-accepted 25% band. Sensitivity tests on 5- and 20-project subsets reveal similar patterns, confirming that the inaccuracy is systemic rather than dataset-specific. Using uncalibrated COCOMO in present-day projects poses a high risk of severe under- or over allocation of resources, potentially trigerring budget overruns and schedule slips. This study quantitatively reveals the limitations of the baseline model, this work provides a benchmark for and a roadmap toward-targeted parameter calibration and hybrid approaches that incorporate additional cost drivers or machine-learning techniques. Future research should explore automatic parameter tuning and context-aware hybrid models to achieve dependable effort estimation in contemporary software engineering.

Keywords: Sofware Effort Estimation, COCOMO, MAE Metric, MMRE, Model Calibration, NASA Projects

1. INTRODUCTION

Accurate software effort estimation is vital for project success, directly impacting budget planning, scheduling, and resouce allocation [1]. Among the various estimation approaches, the Constructive Cost Model (COCOMO), introduced by Boehm in 1981 and later refined in COCOMO II [2], remains a widely referenced algorithmic method. However, its standard parameters are often criticized for lacking accuracy when applied to modern and diverse software projects [3],[4]. Prior studies have explored the limitations of uncalibrated COCOMO and proposed alternatives. [1] Conducted a comparative analysis of traditional and modern estimation techniques, revealing that while COCOMO is useful as baseline estimator, it often suffers from high error rates unless properly calibrated. [3] Compared multiple estimation models and found that COCOMO underperforms for high-variability projects. Similarity, [5] introduced a hybrid approach integrating machine learning with algorithmic estimation, resulting in improved prediction accuracy. [6] Performed a systematic review and comfirmed that machine learning methods though requiring larger and cleaner datasets tend to outperform traditional models. [7] Proposed a context-aware hybrid model using COCOMO II and artificial neural networks, showing better performance for global software development (GSD) projects. While such innovations are promising, there remains a gap in understanding how well the standard, uncalibrated COCOMO model performs on real-world project data. Few empirical studies have focused on measuring its base performance using historical datasets with varying characteristics. This study aims to fill that gap by evaluating the original COCOMO model using standard organic-mode prameters (a = 2.4, b = 1.05) on the COCOMONASA dataset. The model's estimation will be compared to actual effort data using standard metrics Mean Absolute Error (MAE), Mean Magnitude of Relative Error (MMRE), and PRED(0.25). The findings will help clarify the limitations of baseline model and provide a benchmark for future calibration or hybridization efforts in software effort estimation.

2. RESEARCH METHODOLOGY

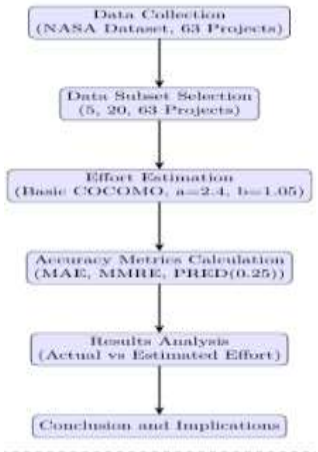

Figure 1. Research Flow

BULLETIN OF INFORMATION TECHNOLOGY (BIT)
Vol 6, No 2, Juni 2025, Hal. 126 - 135
ISSN 2722-0524 (media online)
DOI 10.47065/bit.v5i2.2027
https://journal.fkpt.org/index.php/BIT

This study adopts a quantitve research approach to evaluate the accuracy of basic Constructive Cost Model (COCOMO) in estimating software development effort. The research involves a systematic process consisting of data preprocessing, application of the COCOMO formula, and evaluation using statistical error metrics.

a Dataset Selection
The study uses the cocomonasa_v1 dataset, which constains historical data from 63 software development projects, sourced from PROMISE repository. Each project entry includes the size of the software in Lines of Code (LOC) and the actual effort (in person-months). This dataset is suitable for evaluating estimation models due to its diversity in project sizes and domains.

b Data Processing
To ensure compbility with the COCOMO model, the LOC values are converted into KLOC (Kilo Lines of Code). The processing include cleaning outliers, handling missing value, normalizing size to maintain consistency across records.

c Model Implementation
The basic COCOMO formula is applied to estimate effort:
$Estimated\ effort = a * (Size)^b$ (1)
Equation (1) is:
Size refers to the projects size in KLOC
a = 2.4 and b = 1.05, which are the standard constants for organic type projects. These default parameters are chosen as the focus of this study is evaluate the model's baseline accuracy without any parameter tuning.

d Evaluation Metrics
Three performance metrics are used to assess the estimation accuracy:
Mean Absolute Error (MAE): Measures the average of the absolute differences between estimated and actual effort values.
Mean Magnitude Relative Error (MMRE): Calculates the Average relative deviation. It is a standard metric in software estimation studies but know to be sensitive to extreme values.
Prediction at 25% (PRED(0.25)): Indicates the percentage of estimates where the relative error is less than or equal to 25%, serving as an industry-recognized threshold for practical accuracy.
These metrics provide results are compared to the actual effort values in the dataset. The evaluation metrics are calculated for different subsests of the data (using 5, 20, and all 63 projects) to examine the model's behaviour under varying data volumes.

The results are then analyzed to assess:
Whether the standard COCOMO model performs consistenly across project sizes.
The level deviation from actual effort.
Whether the model can meet acceptable accuracy PRED(0.25). The analysis was performed to evaluate whether the estimates generated with COCOMO default parameters were accurate enough for the dataset used. If the MAE is too high, the researcher will identify the need to develop further estimation models by adapting other parameters or methods to improve the estimate.
It is important to note this research does not involve primary data collection. Instead, it focuses on systematic evaluation of an existing dataset using established analytical procedures. The method prioritizes result-oriented execution rather than proposal-based planning, making it suitbale for empirical software engineering studies.

2.1 Theoritical Studies

The Constructive Cost Model (COCOMO) is an algorithmic model that is used to estimate the effort, time, and cost required in software development projects. The model was initially developed by Barry W. Boehm in 1981 in response to an industry demand for a quantitative method of planning and managing software projects in a more systematic and predictable manner. COCOMO employs an approach predicated on software size (e.g., Kilo Lines of Code/KLOC) and other customization factors that reflect the technical and managerial characteristics of the project [6].
According to [15], the primary advantage of COCOMO is its transparency and empirical basis, with model parameters derived from historical data of hundreds of previously analyzed software projects. This model facilitates a more objective effort estimation process than methods based on intuition or experience. The COCOMO methodology is a software project management approach that considers various factors to enhance its predictability and controllability. These factors include software reliability, team experience, the tools utilized, and the communication complexity. The three modes are as follows:
$Estimated\ effort = a * (Size)^b$ (1)
Equation (1) is Size is the size of the software in thousands of lines of code (KLOC), and a and b are parameters that depend on the project type (organic, semidetached, or embedded).
The term "organic" is employed to denote a software mode that is characterized by a limited number of developers or employees involved in its development. This software mode is frequently implemented, and the developers are typically either homegrown or small-scale in scale. The Embedded mode is utilized for software that exhibits distinct boundaries,

BULLETIN OF INFORMATION TECHNOLOGY (BIT)
Vol 6, No 2, Juni 2025, Hal. 126 - 135
ISSN 2722-0524 (media online)
DOI 10.47065/bit.v5i2.2027
https://journal.fkpt.org/index.php/BIT

demonstrating a high degree of integration with hardware, software, regulations, and intricate operational procedures. The semi-detached mode is situated between the organic and embedded modes. Typically, software in this mode contains up to 300,000 lines of code.

Table 1. COCOMO Model Constant Values

Model	A	B
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.2

The modes delineated in the Table 1, the table function as a reference for the determination of constants. They present a list of constants based on the mode that is employed.
Mean Absolute Error (MAE) is one of the metrics used to measure prediction accuracy in various statistical and machine learning models. MAE calculates the average of the absolute difference between the predicted value and the true value (observed value). This metric gives an idea of how much error occurs in the prediction, with the smaller the MAE value indicates a more accurate model in producing predictions [10].
MAE is often used in the context of quantitative predictions, such as price estimation, demand forecasting, or effort estimation in software development. It has advantages in terms of stability and interpretability, but is sometimes less sensitive to large errors when compared to MSE or Root Mean Squared Error (RMSE), which penalize large errors [17]. Here is the Mean Absolute Error formula:

$$MAE = \left(\frac{1}{n}\right) * \sum |yi - \hat{y}i| \quad (2)$$

Equation (2) is:
N = total number of data or observations
yi = actual value for the i-th observation
ŷi = predicted value for the i-th observation
Σ = sum symbol, summing all values from i = 1 to n. The formula states that MAE is calculated by summing the absolute difference between the actual value (yi) and the predicted value (ŷi) for each observation, then dividing by the total number of observations (n).
MMRE (Mean Magnitude of Relative Error) is the statistical average of all MRE values in a dataset. State that MMRE is used as an indicator of the global performance of the estimation model, even though it is vulnerable to outliers. Formula:

$$MMRE = \frac{1}{n}\sum_{n=i}^{n} MRE_i \quad (3)$$

Equation (3) is:
n = number of projects in the dataset
MREE_i = total accumulated relative error across projects.

PRED(x) calculates the percentage of projects whose estimates meet a certain orrer tolerance. Emphasize that PRED(0.25) is the industry standard for assessing model practicality.
Formula:

$$PRED(x) = \left(\frac{Count(MRE_i \leq x)}{n}\right) x100\% \quad (4)$$

Equation (4) is:
x = Threshold error (usually 0.25 or 25%)
Count (MRE_i ≤ x) = Number of projects with errors below tolerance.

2.2 Research Tools

It is a high-level programming language that is interpreted, dynamic, and supports multiple programming paradigms, including procedural, object-oriented, and functional [23]. The design of the language was informed by a philosophy that placed significant emphasis on enhancing the readability of code through the use of clear and intuitive syntax, as well as the substantial utilization of whitespace for the purpose of establishing code block structures [11]. Python supports automatic memory management with garbage collectors and uses a dynamic typing system that allows flexibility in variable approximation [9]. The Python ecosystem is enriched by the Python Package Index (PyPI), which provides a vast repository of third-party modules and libraries catering to a myriad of computing needs, ranging from web development to scientific computing [13]. The language's cross-platform capabilities enable it to function on multiple operating systems without the need for substantial code modifications [4].

BULLETIN OF INFORMATION TECHNOLOGY (BIT)
Vol 6, No 2, Juni 2025, Hal. 126 - 135
ISSN 2722-0524 (media online)
DOI 10.47065/bit.v5i2.2027
https://journal.fkpt.org/index.php/BIT

Figure 2. Python

A cloud-based computing platform that provides a managed Jupyter Notebook environment for the collaborative writing, execution, and sharing of Python code, thereby obviating the need for local configuration. Colab, a software-as-a-service (SaaS) based service, eliminates the need for local installation by providing instant access to a complete Python environment through a browser. This service leverages Google cloud infrastructure for code execution on temporary virtual machines [5]. The platform has been specifically designed to support scientific computing by providing free access to computing resources such as CPUs, GPUs, and TPUs that have been automatically configured, and it comes with a variety of popular libraries such as TensorFlow, Pytorch, and Pandas ready to use [7],[13]. Recent advancements in technology have enabled the development of real-time collaboration features that facilitate simultaneous editing of notebooks by multiple users, akin to the functionality observed in Google Docs. These features are further enhanced by integrations with prominent cloud storage platforms such as Google Drive and GitHub, which offer efficient data storage and version control mechanism [7].

Figure 3. Google Colaboratory

3. RESULT AND DISCUSSION

The dataset used in this analysis is "cocomonasa" which contains data from 63 software projects. This dataset includes attributes such as Lines Of Code (LOC), Actual Effort (ACT_EFFORT), and several other attributes that are not used in this analysis. The range of LOCs in a dataset varies, ranging from 2,000 to 100,000 lines of code. This dataset is assumed to represent projects with "organic" characteristics according to the COCOMO model. The data source is available in the Promise (http://openscience.us/repo/) software engineering data repository.

Table 2. Dataset

LOC	ACT_EFFORT
70	278
227	1181
177.9	1248
115.8	480
29.5	120
19.7	60
66.6	300
5.5	18
10.4	50
14	60
16	114
6.5	42

BULLETIN OF INFORMATION TECHNOLOGY (BIT)
Vol 6, No 2, Juni 2025, Hal. 126 - 135
ISSN 2722-0524 (media online)
DOI 10.47065/bit.v5i2.2027
https://journal.fkpt.org/index.php/BIT

13	60
8	42
90	450
15	90
38	210
10	48
161.1	815
48.5	239
32.6	170
12.8	62
15.4	70
16.3	82
35.5	192
25.9	117.6
24.6	117.6
7.7	31.2
9.7	25.2
2.2	8.4
3.5	10.8
8.2	36
66.6	352.8
150	324
100	360
100	215
100	360
15	48
32.5	60
31.5	60
6	24
11.3	36
20	72
20	48
7.5	72
302	2400
370	3240
219	2120
50	370
101	750
190	420
47.5	252
21	107
423	2300
79	400
284.7	973
282.1	1368
78	571.4
11.4	98.8
19.3	155

BULLETIN OF INFORMATION TECHNOLOGY (BIT)
Vol 6, No 2, Juni 2025, Hal. 126 - 135
ISSN 2722-0524 (media online)
DOI 10.47065/bit.v5i2.2027
https://journal.fkpt.org/index.php/BIT

3.1 Mean Absolute Error (MAE)

The results show an MAE value of 661. This MAE value indicates the average absolute difference between the estimated effort column generated by the COCOMO model and the actual effort of the 5 projects in the dataset.Tests were conducted using 20 data points. At Table 3 the previous MAE test results only used 5 data points, the latest test results show an MAE value of 295. This new MAE value indicates that the average absolute difference between the effort estimates generated from the 20 projects in the dataset is 295 person-months. With an MAE of 295, the COCOMO model still has a significant prediction error. However, this error is lower than the MAE of 661.12. The comparison of MAE values using different data shows an interesting pattern. The MAE for 5 data is 661, indicating a fairly high average absolute error in these early projects. However, the MAE decreased significantly to 295 when evaluated on 20 data. This decrease indicates that the addition of projects from the 6th to the 20th data helps to reduce the average estimation error. Interestingly, when the entire dataset of 63 projects is used, the MAE again increases to 406. This pattern suggests that projects beyond 20 data tend to be more difficult to predict by the basic COCOMO model compared to the 6-20 data subset, although the average error for the entire dataset is still lower than the average error at 5 data.

Table 3.MAE Results

MAE (5 DATA)	661.129
MAE (20 DATA)	295.127
ALL DATA	406.25

3.2 Mean Magnitude Relative Error (MMRE)

The MMRE value obtained is 0.9995907211638551 MMRE represents the average percentage error of the estimated effort. The MMRE value of 1.0 indicates that, on average, the COCOMO model misses about 100% of the actual effort. This indicates that the model has a relatively large error compared to the actual effort and needs to be improved. In Table 4, the test results using the number of data 5 to 20, show an increase as the data data points used increases, the Mean Magnitude of Relative Error (MMRE) value increases from 0.9995907211638551 to 0.995863150932857. While more data should provide more information for the model, in this case, the additional data actually caused the model to produce missed predictions, thus increasing the MMRE. Evaluation using MMRE, which considers the error relative to the project size, shows a different result. The MMRE value of 0.9996, for the initial 5 data points indicates that, on average, the model error nearly equals the actual effort- suggesting poor estimation accuracy. The MMRE for 20 data is 0.9996. Referring to the decrease in MAE from 661 to 295 indicates that the average magnitude of estimation error is reduced. However, the MMRE remains high indicating that while this error may be smaller in absolute terms, it is still a large percentage of the actual project effort. Although the code comparison shows a decrease in MMRE with the addition of data, numerically the decrease is very small (0.9995907..) to (0.9995863..) and not substantially significant. This high stability of MMRE, as opposed to the significant decrease in MAE, suggests that although the average absolute error may be reduced with the use of 20 data points, it is still a very large proportion of the actual project effort.

Table 4. MMRE Results

MMRE (5 DATA)	0.9995907
MMRE (20 DATA)	0.9995863

3.3 Prediction at Level (PRED(0.25))

The analysis shows a PRED(0.25) value of 0.0, indicating that the COCOMO model has very low accuracy for the dataset used. This is due to the characteristics of the projects in the dataset that do not match the assumptions of the COCOMO model, or because the model parameters are not optimized. To improve the estimation accuracy, it is recommended to tune the model parameters or try other more suitable effort estimation models.
The PRED(0.25) metric is designed to measure how often the model provides predictions that fall within the error range considered ‘acceptable’ in many effort estimation contexts (25% is often used as a threshold). For the test results PRED remained constant at 0.0 even though the amount of data used in the COCOMO model was increased from 5 data points to 20 data points. This indicates that no project, whether using 5 or 20 datasets, has a relative error (MRE) less than or equal to 25%. In other words, the COCOMO model consistently produced effort estimates that missed more than 25% of the actual effort for all projects in the tested dataset, both with 5 and with 20 data points. Adding 15 new projects or data to the evaluation did not increase the proportion of projects predicted within the 25% limit. This is consistent with the MMRE findings, which remain high, indicating that the relative error is generally very large, far exceeding 25%. A value of 0.0 indicates that the basic COCOMO model with default parameters is completely unable to produce estimates that are within 25% of the actual effort for projects from 20 data points.

BULLETIN OF INFORMATION TECHNOLOGY (BIT)
Vol 6, No 2, Juni 2025, Hal. 126 - 135
ISSN 2722-0524 (media online)
DOI 10.47065/bit.v5i2.2027
https://journal.fkpt.org/index.php/BIT

Table 5. PRED Results

PRED(0.25) (5 DATA)	0.0
PRED(0.25) (5 DATA)	0.0
PRED(0.25) (ALL DATA)	0.0

3.4 Visualization

Figure 4 shows presents a scatter plot that visually compares Actual Effort with Estimated Effort. These values are based on the implementation of an estimation model. The X-axis of the graph indicates the Actual Effort value, while the Y-axis denotes the Estimated Effort value. The red dotted line that intersects the plot signifies the ideal condition, defined as the point at which the Estimated Effort Value is equivalent to the Actual Effort.

According to the visual representation, the majority of the data points are situated well below the y = x line. This finding suggests that the COCOMO model tends to generate estimates that underestimate the actual effort expended in software projects. It is noteworthy that certain points exhibit pronounced disparities, with actual effort surpassing 1,000 units, while estimated effort persists within the minimal range, specifically from 500 to nearly zero.

These findings suggest that the accuracy of effort estimation using the COCOMO model in this study is relatively low, and further evaluation of the model parameters and the estimation approach used is necessary. Adjustments to the model parameter values or the integration of additional correction factors have been identified as possible solutions to improve estimation accuracy.

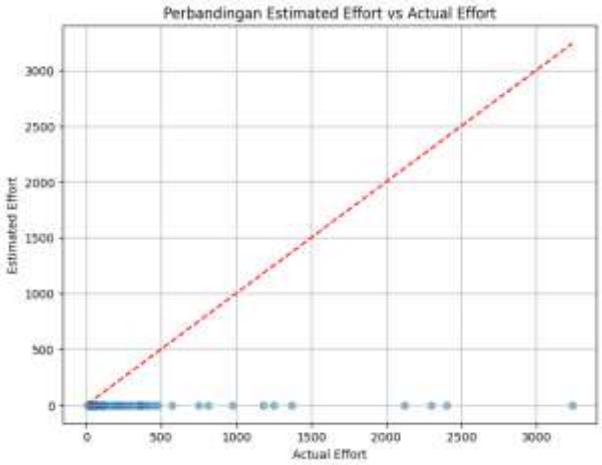

Figure 4. Comparison Estimated vs Actual Effort

3.5 Analysis

Based on the results of the metric evaluation, the basic COCOMO model with basic organic parameters showed that the performance used was unsatisfactory, in estimating absolute error (MAE) for the entire dataset reaching 406 man-months. Furthermore, the mean relative error (MMRE) for the 20 data is very high at 0.9996, indicating that the average estimation error is almost as large as the actual effort of the project itself.

MMRE tends to increase when using 20 amounts of data compared to using 5 amounts of data, and then return almost the same value when using the entire dataset. This suggests that the COCOMO model tends to produce more estimates that are more relatively missed when faced with more data even though too significant.

A very high MMRE (1.00) indicates that, on average, the COCOMO model misses about 100% of the actual project effort. This means that the estimate generated by the model can be double or even greater than the actual effort. This shows that the COCOMO model is not able to provide estimates that are close to the actual value.

The constant PRED(0.25) value at 0.0 for all data scenarios indicates that the COCOMO model consistently produces highly missed estimates for all projects in the dataset, unaffected by the amount of data used.

Based on the results of the analysis, it can be concluded that the COCOMO model has low accuracy in estimating the effort for the dataset used. Although there was a slight decrease in MAE by 20 data, this was enough to offset the overall poor performance of the model, especially in terms of MMRE.

3.6 Comparison and Results

Mean Absolute Error (MAE): The initial five data points were evaluated, yielding an MAE of 661. This value indicates a relatively high average absolute estimation error in the initial project subset. The quantity of data has been augmented to 20, while the mean absolute error (MAE) has undergone a substantial decline, reaching 295. This decline, amounting to nearly 50%, suggests that incorporating projects from the sixth to the twentieth data point tends to yield a relative estimation error that is comparatively smaller than that associated with the initial five projects. This could be due to the specific characteristics of the first five projects, which make them more difficult to predict by the basic model, or

BULLETIN OF INFORMATION TECHNOLOGY (BIT)
Vol 6, No 2, Juni 2025, Hal. 126 - 135
ISSN 2722-0524 (media online)
DOI 10.47065/bit.v5i2.2027
https://journal.fkpt.org/index.php/BIT

the presence of outliers at the beginning of the dataset.b) Mean Magnitude of Relative Error (MMRE): The mean relative error (MRE) for the initial five data points is 0.9996. This value approaches 1, suggesting that, on average, the absolute estimation error is nearly equivalent to the actual project effort, indicating a disproportionately poor performance ratio. Initially, the MMRE was determined to be 0.9996, based on the initial 20 data points. While the comparison code indicates that MMRE decreases with additional data, the numerical decrease from 0.9995907 to 0.9995863 is negligible and not practically significant. The stability (or very minimal decrease) in this very high MMRE value, despite the initial significant decrease in MAE, is an important finding. The findings indicate that the supplementary projects within the 6-20 subset, despite potentially exhibiting reduced absolute errors as evidenced by the mean absolute error (MAE) metric, characteristically involve smaller project sizes.
Prediction Level at 0.25 (PRED(0.25)) The initial five data points are indicated as 0.0. Consequently, it can be deduced that none of the initial five projects exhibit a relative estimation error of 25% or less.
According to the comparison code result, the value of RED remains constant at 0.0, even in the presence of additional data points. This observation persists for the initial 20 data points. This outcome is of significant concern. The absence of enhancement in PRED upon elevating the data to 20 data points signifies that the baseline COCOMO model exhibited an inability to generate estimates that satisfied the 25% tolerance range criteria for this baseline. Despite the fluctuations in MAE, the model was never "close enough" to the actual effort in proportions deemed acceptable.

3.7 Practical Implications

The findings of this study carry significant implications for real-world software project estimation. In practice, project managers often rely on baseline models like COCOMO due to their simplicity and historical use. However, the results here demonstrate that using the standard COCOMO model without any calibration can lead to severe underestimation-with average errors approaching 100% and No. predictions falling within the acceptable 25% error range. For current software egineering practice, this highlights a major risk: project stakeholders who use outdated or uncalibrated models may inavertenly make critical planning decisions based on flawed assumptions. This can cause cascading effects, such as resource shortages, unmet deadlines, and project failures. As such, practitioners are advised to avoid direct application of COCOMO in its original form and instead consider calibrating the model using historical project data or adopting hybrid models that integrate algoritmic and machine learning-based approaches. Furthermore, this study provides a benchmark reference for organizations looking to evaluate the fitness of classical estimation models in their project environments.

3.8 Discussion

This research was conducted inseparable from the results of previous studies that been carried out as material for comparison and study. The research results that are used as comparisons cannot be separated form research topic.

1. Effort estimation in software development has become a critical topic of interest in recent years due to its direct impact on cost efficiency, project planning, and resource allocation. Various approaches have been developed, including algorithmic models such as the Constructive Cost Model (COCOMO), which remains one of the most widely cited methods
2. [1] conducted a comparative analysis between traditional and modern estimation techniques. Their findings indicate that while COCOMO remains useful as a baseline estimator, it often produces high error rates if not calibrated properly to the dataset or project context.
3. [14] reviewed machine learning based effort estimation methods and found that techniques such as regression, neural networks, and assemble models generally outperform classical algorithmic models. Nevertheless machine learning methods require large, high-quality datasets and often lack explainability.
4. [3] proposed hybrid models that combine algorithmic estimation techniques with machine learning based-parameter tuning to improve predictive accuracy. Their results demonstrate the potential of adaptive methods in handling dataset variability.
5. [3] compared various estimation models, including COCOMO, SLIM, and regression-based approaches. This study found that COCOMO struggled to deliver accurate predictions for projects with high variability, and they recommended the adoption of adaptive or hybrid techniques.
6. Overall, these studies consistently highlight that while COCOMO is advantageous for its transparency and ease of implementation, it often lacks the accuracy required for modern projects unless it is recalibrated or combined with more adaptive techniques. Further empirical evaluation of the baseline COCOMO models is still relevant to determine its standalone accuracy in modern datasets.
7. Recent research futher emphasizes the move towards hybrids and context-aware models. For instance, [2] developed a hybrid model combining COCOMO II with Atificial Neural Network (ANN), specifically incorporating cost drivers relevant to Global software Development (GSD). Their findings suggest that such hybrid approaches, tailored to specific development contexts like GSD, can significantly outperform traditional models by addressing limitations related to historical data reliance and subjectivity [2].

4. CONCLUSION

BULLETIN OF INFORMATION TECHNOLOGY (BIT)
Vol 6, No 2, Juni 2025, Hal. 126 - 135
ISSN 2722-0524 (media online)
DOI 10.47065/bit.v5i2.2027
https://journal.fkpt.org/index.php/BIT

The Findings of this study demonstrate that the standard COCOMO model, when applied without calibration, produces high estimation errors across various project sizes. With Mean Absolute Error (MAE) values ranging from 295 to 662 and a Mean Magnitude of Relative error (MMRE) approaching 1.0, the model exhibits significant deviation from actual effort values. Furthermore, a PRED(0.25) value of 0.0 indicates that none of the predictions fell within an acceptable 25% error threshold.The results directly address the research problem regarding the accuracy of the baseline COCOMO model and confirm the identified gap in the literature: the lack of empirical evaluation of the uncalibrated model on real-world datasets. The performance of model suggests that its fixed parameters are not generalizable to diverse modern project contexts.These result confirm that while COCOMO offers a structured and interpretable estimation approach, its practical application without calibration leads to consistenly inacurrate predictions. For practitioners, this implies that relying on default parameters may result in suboptimal planning. Incorporating domain-specific tuning or leveraging adaptive estimation models is necessary to align prediction accuracy with the complexity of modern software development projects.In addition to reaffirming the limitations of the uncalibrated COCOMO model, this study offers actionable insights for future model improvements. The consistently high MMRE and MAE values cross diverse project sizes suggest that the static parameters (a = 2.4, b = 1.05) used in the organic mode are not suitable for modern software development contexts. Therefore, future research could explore dynamic parameter tuning using regression or optimization techniques such as genetic algorithms, particle swarm optimization, or Bayesian calibration. Another promising direction involves hybriding COCOMO with machine learning models. For instance, machine learning could be used to classify project types and predict appropriate COCOMO parameters based on project metadata. Alternatively, the integration of contextual factors such as team experience, tool maturity, or domain complexity into the model could improve estimation realism. These enchancements would enable more accurate and adaptable models suitable for real-world application, thereby advancing the field of software effort estimation.

REFERENCES

[1] M. Ahmad and M. A. Wani, "Comparative Analysis of Traditional and Modern Software Effort Estimation Techniques," *Int. J. Comput. Appl.*, vol. 175, no. 3, pp. 22–27, 2023.
[2] M. Ahmed, N. B. Ibrahim, A. W. Ahmed, M. Juniadi, E. S. Flores, and D. Anand, "A Hybrid Model for Improving Software Cost Estimation in Global Software Development," *Comput. Mater. Contin.*, vol. 78, no. 1, pp. 1399–1422, 2024. [Online]. Available: https://doi.org/10.32604/cmc.2023.046648
[3] M. Alenezi and A. K. Mahmood, "A Comparative Study of Software Effort Estimation Models," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 8, pp. 1–6, 2020.
[4] D. Beazley, *Python Essential Reference*, 5th ed. Boston, MA: Addison-Wesley, 2020.
[5] E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Berkeley, CA: Apress, 2019.
[6] B. W. Boehm, C. Abts, and S. Chulani, *Software Cost Estimation with COCOMO II*. Upper Saddle River, NJ: Prentice Hall, 2020.
[7] F. Chollet, *Deep Learning with Python*, 2nd ed. Shelter Island, NY: Manning Publications, 2021.
[8] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 3rd ed. Sebastopol, CA: Apress, 2019.
[9] J. V. Guttag, *Introduction to Computation and Programming Using Python*, 3rd ed. Cambridge, MA: MIT Press, 2021.
[10] M. Jørgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Trans. Softw. Eng.*, vol. 45, no. 1, pp. 1–26, Jan. 2019. [Online]. Available: https://doi.org/10.1109/TSE.2017.2754298
[11] M. Lutz, *Learning Python*, 5th ed. Sebastopol, CA: O'Reilly Media, 2019.
[12] Y. Mahmood *et al.*, "Software Effort Estimation Accuracy Prediction of Machine Learning Techniques: A Systematic Performance Evaluation," *arXiv preprint arXiv:2101.10658*, 2021. [Online]. Available: https://arxiv.org/abs/2101.10658
[13] W. McKinney, *Python for Data Analysis*, 3rd ed. Sebastopol, CA: O'Reilly Media, 2020.
[14] S. Pandey and S. Brave, "Software Effort Estimation Using Rao Algorithm," *Int. J. Inf. Eng. Electron. Bus.*, vol. 11, no. 1, pp. 16–24, 2019.
[15] S. Pandey, S. K. Dubey, and A. Rana, "Software effort estimation techniques: A systematic literature review," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 33, no. 1, pp. 10–25, 2021. [Online]. Available: https://doi.org/10.1016/j.jksuci.2018.03.007
[16] M. Q. Patton, *Utilization-Focused Evaluation*, 5th ed. Thousand Oaks, CA: Sage Publications, 2020.
[17] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. New York: McGraw-Hill Education, 2020.
[18] S. Raschka and V. Mirjalili, *Python Machine Learning*, 3rd ed. Birmingham, UK: Packt Publishing, 2020.
[19] S. A. Raza, M. R. J. Qureshi, and F. Ahmad, "Software Effort Estimation using Machine Learning Techniques: A Systematic Literature Review," *J. Softw. Evol. Process.*, vol. 34, no. 2, e2384, 2022.
[20] I. Sommerville, *Software Engineering*, 10th ed. Boston, MA: Pearson Education, 2020.

BULLETIN OF INFORMATION TECHNOLOGY (BIT)
Vol 6, No 2, Juni 2025, Hal. 126 - 135
ISSN 2722-0524 (media online)
DOI 10.47065/bit.v5i2.2027
https://journal.fkpt.org/index.php/BIT

[21] D. L. Stufflebeam and C. L. S. Coryn, *Evaluation Theory, Models, and Applications*, 2nd ed. San Francisco, CA: Jossey-Bass, 2019.
[22] D. Thakur and G. Kaur, "A Review of Software Effort Estimation Techniques," *Int. J. Sci. Technol. Res.*, vol. 10, no. 1, pp. 258–262, 2021.
[23] G. van Rossum, "Python Programming Language," Python Software Foundation, 2020. [Online]. Available: https://www.python.org/
[24] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Commun. ACM*, vol. 64, no. 3, pp. 107–115, 2020. [Online]. Available: https://doi.org/10.1145/3446776