Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 497-510
https://doi.org//10.47514/kjcs/2024.1.3.007

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

An Enhanced COCOMO II Model to Function Point Analysis to Estimate Effort, Duration, and Productivity using a Modified Backfiring Algorithm in Software Development

Philip O. Odion and Hamza Umar.
Department of Computer Science, Nigerian Defence Academy, Kaduna, Nigeria.
poodion@nda.edu.ng, hamzaumar9@nda.edu.ng

ABSTRACT

Accurate and reliable software effort and cost of the software are very important for project managers in planning and scheduling the software in the software development industry. While backfiring is useful and simple, there is a high margin of error in converting SLOC data into function points. Comparing two most widely used software cost estimation models Function Point Analysis and COCOMO II model can yield different result and error. Modification of the backfiring algorithm is essential for improving the accuracy and reliability of software development effort estimation. This study presented a hybrid model combining Function Point Analysis (FPA) and the COCOMO II framework to enhance estimation accuracy. The study involves developing a user-friendly interface where users input project metrics, and the system computes Unadjusted Function Points (UFP), Complexity Adjustment Factors (CAF), Adjusted Function Points (AFP), and convert Source Lines of Code (SLOC) to FP using a modified Backfiring Algorithm and vice versa. The proposed model's performance has been evaluated using datasets from Turkish, Industry, and NASA COCOMO dataset, demonstrating its robustness and generalizability. The results indicate that the hybrid model significantly improves effort estimation accuracy compared to traditional methods. Performance metrics, including Mean Absolute Error (MAE) and Mean Magnitude Relative Error (MMRE), show that the proposed model provides estimates closer to actual effort, improving accuracy by 9.5%.. Cross validation confirms the significance of these improvements. The findings suggest that the Enhanced model can better inform project managers, reducing cost overruns and improving project timelines. Limitations of the study include the diversity and size of the datasets; which future research could address by incorporating more varied data. Additionally, integrating advanced machine learning techniques could further enhance the model's predictive capabilities. This dissertation contributes to the field of software engineering by offering a practical and improved approach to software effort estimation, supporting more accurate and efficient project management.

Keywords: *Software Cost Estimation, Function Point, COCOMO II, Backfiring Algorithm, Unadjusted Function Point.*

1. Introduction

Software estimation is a critical issue in the development of a software system. This problem has grown into a crucial question since numerous projects continue to not be completed on time - with either under or overestimation of efforts resulting in specific issues. For this reason, different software cost estimation models were developed to manage the budget and schedule of software projects. Accurate software effort estimations (SEEs) are essential to project managers, as well as customers. It supports the generation of applications for proposals, contract negotiations, scheduling, monitoring, and control (Hai et al., 2019).

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 497-510
https://doi.org//10.47514/kjcs/2024.1.3.007
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Software Effort Estimation (SEE) focuses on how to estimate the effort, time, and cost with the project activities schedule. In SEE, two types of conventional methods are typically used to quantify the impression of non-algorithmic procedures before settling on an algorithmic approach to dealing with proving human science. The algorithmic cycle is produced by doing the product evaluation utilizing some mathematical demonstrations. These logical, numerical models are based on the recorded data and make use of information sources including size, the number of features and capacities, and other expense factors. Among the models that make use of the algorithmic methodology are the Putnam Model, the Function Point Analysis (FPA) model, and the Constructive Cost Model (COCOMO) model (Purushotham & Kumar, 2022).

The modern day software industry is all about efficiency. With increase in expanse of software projects, it has become important to analyze the software requirements early in the software development phase. For a given set of requirements, it is desirable to evaluate the amount of time and the money required for delivering the project prolifically (Kaur & Sehra 2014). Accurate estimation of the problem size is very important to obtain a satisfactory estimation of effort, cost and time duration of a software project. The size of the problem is not the number of bytes that the source code occupies or the byte size of the executable code. But it is a measure of the problem complexity in terms of the effort and time required to develop the product. Two widely used metrics to estimate the size are: Lines of Code (LOC) and Function Points (FP) (Kaur & Sehra 2014). The accurate and reliable effort and cost of the software are very important for project managers in planning and scheduling, particularly in the early stages of the software development life cycle (SDLC). Any improvement in the accuracy of predicting the development effort can significantly reduce the cost arising from errors (Frank, 2019). Reasons for Software Estimation involves answering these questions; how much effort is required to complete each activity, how much calendar time is needed to complete each activity and what is the total cost of each activity? (Odion & Onibere, 2014). Some available software cost estimation models are shown in Figure 1.

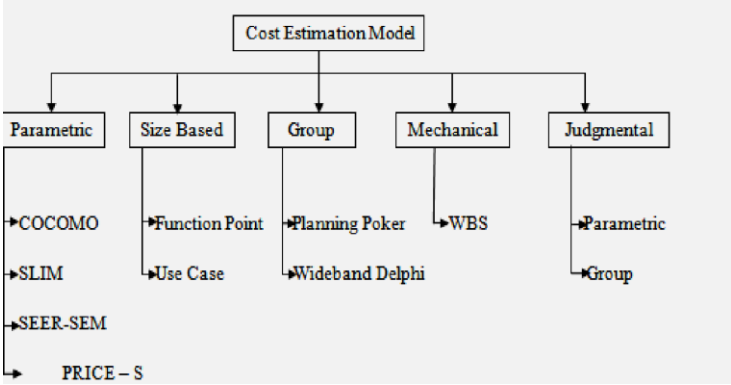

Figure 1: Estimation Models (Kaur & Singh, 2015).

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 497-510
https://doi.org//10.47514/kjcs/2024.1.3.007
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Function Point Analysis was developed by Allan J. Albrecht in 1979. Function points are independent of language, technology, tools and database. Function point is a unit of measurement for determining the functional size of an information system. Function point analysis is a process
that involves identifying major system components and classifying them as 'simple', 'average' or 'complex' (Wong et al., 2016). Function point estimates the size of a software project using five elements: Internal Logical Files (ILF), External Interface Files (EIF), External Inputs (EI), External Outputs (EO) and external Enquiries (EQ). Function point calculations begin with counting the five elements. Each function point element is assigned a complexity level (Low, Average, High) based on its associated file number. The associated file numbers are described as Data Element Type (DET), File Type Referenced (FTR) and Record Element Types (RET)(Uzzafer, 2016). Table 1 shows Function Point Complexity weights used to calculate Unadjusted Function Point (UFP).

Table 1: Function Point Complexity Weights

Component	Low	Average	High
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

Unadjusted Function Point (UFP) is the total number function points counted together. The unadjusted function point is computed from the following equation.

$$UFP = \sum_{i=1}^{5} Wij \times \sum_{j=1}^{3} Xij \quad (2.1)$$

Where w_{ij} is the complexity weight and x_{ij} is the count of each function element. UFP is then multiplied by the Value Adjustment Factor (VAF) to get the function point (FP) count using equation 2.2. The VAF is calculated from 14 General System Characteristics (GSC);Data communications,Distributed data processing,Performance, Heavily used configuration, Transaction rate, On-Line data entry, End-user efficiency, On-Line update,Complex processing, Reusability, Installation ease, Operational ease, Multiple sites, and Facilitate change.

The GSC values are summed to calculate the VAF. Each of the general system characteristics will be assigned a value from 0 to 5 to show its degree of influence. The values of the degree of influence represent: 0 = Not present, or no influence when present. 1 = Insignificant influence. 2 = Moderate influence. 3 = Average influence. 4 = Significant influence.5 = Strong influence at all development stages.

$$VAF = 0.65 + (0.01 \sum_{i=1}^{14} Ci) \quad (2.3)$$

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 497-510
https://doi.org//10.47514/kjcs/2024.1.3.007
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Where C_i are the GSC values. Finally, the UFP and VAF are multiplied to get the function point (FP) count (Uzzafer, 2016).

$$FP = UFP \times VAF \quad (2.4)$$

COCOMO II is the update of Intermediate COCOMO, COCOMO II accounts for recent changes in software engineering technology. The overall COCOMO framework remained about the same but significant changes were found to be necessary to keep pace with the changing nature of software development and evolution. Boehm et al. (2000) demonstrate that, these changes have included a move away from the Waterfall process model towards evolutionary, incremental, and spiral models; product line management approaches to software reuse; application capabilities; and graphics user interface builder tools that made traditional size metrics such as source line of code (SLOC) inappropriate.

In COCOMO II effort is expressed as a function of program size, set of cost drivers, scale factors, Baseline Effort Constants and Baseline Schedule Constants. COCOMO II has some special features, which distinguish it from COCOMO 81. The Usage of this method is very wide and its results usually are accurate. The COCOMO II effort estimation model is formulated as in equation (2.24). This model is used for both early design and post-architecture models to estimate effort. The inputs are the size of software development, a constant *a*, an exponent *E*, and a number of Effort Multipliers (EM). The number of effort multipliers depends on the model being used (Pauline et al., 2013).

$$PM = a \times SIZE^E \times EM \quad (2.24)$$

Where the constant a = 2.94. The exponent *E* is an aggregation of five scale factors. All scale factors have rating levels. These rating levels are Very Low (VL), Low (L), Nominal (N), High (H), Very High (VH) and Extra High (XH). Each rating level has a weight W, which is a quantitative value used in the COCOMO II model. EM is an effort multipliers.

$$\text{Where } E = B + 0.01 \sum_{j=1}^{5} SFj \quad (2.25)$$

Where B = 0.91. B is a constant derived from finished projects. Scale Factors (SF) is a six point ordinal scale ranging from low to high importance. Table 2 shows COCOMO II Scale Factors rating Values.

Table 2: COCOMO II Scale factors rating values.

DRIVER	VL	L	N	H	VH	XH	Productivity Range
PREC	6.20	4.96	3.72	2.48	1.24		1.33
FLEX	5.07	4.05	3.04	2.03	1.01		1.26
RESL	7.07	5.65	4.24	2.83	1.41		1.39

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 497-510
https://doi.org//10.47514/kjcs/2024.1.3.007
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

TEAM	5.48	4.38	3.29	2.19	1.10		1.29
PMAT	7.8	6.24	4.68	3.12	1.56		1.43

Table 3 shows COCOMO II 17 cost drivers used to calculate effort adjustment factors.

Table 3: COCOMO II 17 Cost Drivers (Effort Multipliers) rating values.

DRIVER	VL	L	N	H	VH	XH	Productivity Range
RELY	0.82	0.92	1.00	1.1	1.26	-	1.54
DATA	-	0.90	1.00	1.14	1.28	-	1.42
CPLX	0.73	0.87	1.00	1.17	1.34	1.74	2.38
RUSE	-	0.95	1.00	1.07	1.15	1.24	1.31
DOCU	0.81	0.91	1.00	1.11	1.23	-	1.52
TIME	-	-	1.00	1.11	1.29	1.63	1.63
STOR	-	-	1.00	1.05	1.17	1.46	1.46
PVOL	-	0.87	1.00	1.15	1.3	-	1.49
ACAP	1.42	1.19	1.00	0.85	0.71	-	2.00
PCAP	1.34	1.15	1.00	0.88	0.76	-	1.76
PCON	1.29	1.12	1.00	0.9	0.81	-	1.51
APEX	1.22	1.1	1.00	0.88	0.81	-	1.51
PLEX	1.19	1.09	1.00	0.91	0.85	-	1.40
LTEX	1.20	1.09	1.00	0.91	0.84	-	1.43
TOOL	1.17	1.09	1.00	0.9	0.78	-	1.50
SITE	1.22	1.09	1.00	0.93	0.86	0.80	1.53
SCED	1.43	1.14	1.00	1.0	1.00	-	1.43

While backfiring is useful and simple, there is a high margin of error in converting SLOC data into function points (Wong et al.,2008). Comparing two most widely used software cost estimation models Function Point Analysis and COCOMO II model can yield different result and error. Function Point Analysis focuses on the effort needed for a project and its size in source lines of code (SLC), while COCOMO II can provide cost estimates, duration and productivity. For more accurate software cost estimation, the two methods

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

should be used together to achieve the best results in estimating software cost and reduce the error when the programming language is known.

The objective of this research is to modify the backfiring algorithm to convert COCOMO II model to Function Point Analysis to calculate effort, duration and productivity using the modified model.

This research work would enhance the accuracy, speed, and efficiency of costing Software Development Projects. Through the development of an algorithm and a hybrid model of Function Point Analysis and the COCOMO II model, this research project can help to improve the accuracy of effort, duration, and productivity estimations. This research has the potential to revolutionize the way software projects are managed and deliver much better outcomes for clients.

2. Related Works

Several studies have explored the use of machine learning techniques for software cost estimation. Rahman et al. (2023) proposed the use of machine learning techniques for software effort estimation. The authors argue that machine learning can be used to overcome the limitations of traditional estimation techniques, such as the COCOMO model, which are often inaccurate and time-consuming to use. The authors evaluated three machine learning algorithms for software effort estimation: k-nearest neighbor regression, support vector regression, and decision trees. They used a dataset of software projects from a software company called Edusoft Consulted LTD to train and evaluate the algorithms. The results of the study showed that decision trees performed the best in terms of accuracy, followed by k-nearest neighbor regression and support vector regression. The authors concluded that decision trees are a promising machine learning algorithm for software effort estimation. Khan et al. (2021) presented an improved version of the COCOMO-II model that incorporates several additional cost drivers specific to global software development (GSD) context. The authors aimed to enhance the accuracy of cost estimation in GSD projects, which is a significant challenge due to the geographical, cultural, and linguistic diversity of teams. The results of the study demonstrated that the Amplified COCOMO-II model outperformed the original COCOMO-II model in terms of cost estimation accuracy for GSD projects. The authors discussed the practical implications of their model, including its potential use by project managers to make informed decisions regarding project budgeting, resource allocation, and team formation. Limitations of their study include the small sample size and the use of self-reported data. Manikavelan and Ponnusamy (2020) in a paper titled "Software quality analysis based on cost and error using fuzzy combined COCOMO model" presented a fuzzy-COCOMO model for software quality analysis. The model combined the COCOMO II cost estimation model with fuzzy logic to predict the quality of software projects. The model uses project size, type, and cost drivers to estimate the effort, schedule, and cost of a software project. The model also uses fuzzy logic to estimate the quality of the software project. The authors evaluated the model using data

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

from 20 software projects. The results showed that the model was able to predict the quality of the software projects with an accuracy of 95%. The authors also found that the model was able to identify the factors that had the greatest impact on the quality of the software projects.

Odion & Alfa (2022) in a paper titled " An Optimal Software Development Process Tasks Sharing Techniques Using Effort Multiplier Values of Software Project Management" addressed the challenge of inefficient task allocation in software development. Their research developed an optimal task sharing system using the COCOMO 81 model to improve project performance. By considering developer and task attributes, the system effectively allocated tasks, resulting in reduced project timelines and costs. This study contributes to the field by demonstrating the practical application of software estimation techniques for enhancing task management in software development. Accurate software cost estimation is a critical factor in successful project management. Several models, such as COCOMO II, have been developed to predict project costs. However, the accuracy of these models can be influenced by various factors. Putri et al. (2024) proposed a modified Hunting Gray Wolf Optimization (HGWO) algorithm to enhance COCOMO II's performance in a paper titled" COCOMO II-FG-HGWO: A Modified Hunting Gray Wolf Optimization for Improving Constructive Cost Model II Performance.". By incorporating optimization techniques, the authors aimed to improve the precision of cost estimates. This study contributes to the growing body of research on applying metaheuristic algorithms to software engineering challenges. Ahmed et al., (2024) in a paper titled "A Hybrid Model for Improving Software Cost Estimation in Global Software Development" Introduced a novel hybrid model that synergizes the COCOMO II with Artificial Neural Networks (ANN) to address the challenges of software cost estimation in Global Software Development (GSD). Traditional models like COCOMO II, while widely used, often struggle to capture the complexities of global software development. To address these limitations, researchers have explored various approaches, including the use of optimization techniques and hybrid models combining COCOMO II with machine learning. The study builds upon these efforts by incorporating a comprehensive set of GSD-specific cost drivers into a hybrid model, aiming to improve estimation accuracy and provide valuable insights for project managers. Gupta et al. (2024), in their paper titled "Software Cost Estimation: A Comparative Analysis," discussed the process of determining the most accurate approach for software cost estimation to achieve higher levels of accuracy. Software cost

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 497-510
https://doi.org//10.47514/kjcs/2024.1.3.007
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

estimation is one of the most crucial aspects of software development, involving the calculation of the time and money needed to complete software projects. Numerous approaches have been developed and implemented for estimating software costs, each with its own strengths and limitations. Appropriate cost estimation ensures the timely and cost-effective completion of software projects. However, the authors concluded that no single strategy works well in every situation or for every project. Therefore, a comprehensive understanding and rigorous analysis are needed to compare all approaches and determine the best one likely to produce realistic and accurate estimations.

Bajusova et al., (2024) in a paper titled "Enhancing Software Effort Estimation with Self-Organizing Migration Algorithm: A Comparative Analysis of COCOMO Models" conducted an in-depth analysis to improve the accuracy of software effort estimation using a Constructive Cost Model (COCOMO) optimized with the Self-Organizing Migration Algorithm (SOMA). The research focused on key evaluation metrics such as Mean Magnitude of Relative Error (MMRE), Prediction at 0.25 (PRED(0.25)), Mean Absolute Error (MAE), and Root Mean Square Error (RMSE). The study covered various COCOMO model configurations—basic, intermediate, and post-architecture COCOMO II—and included additional statistical tests and residual analysis for detailed insights. The findings indicated that SOMA-optimized COCOMO models generally outperformed traditional models in predictive accuracy, with improvements observed in several metrics: up to 12% in MMRE, a 15% increase in PRED(0.25), an 18% reduction in MAE, and a 20% decrease in RMSE. Due to the considerable margin of error in conventional estimating methods and the development of machine learning algorithms that can increase forecast accuracy, data mining has become increasingly popular in software estimation (Antil et al., 2023). While backfiring is useful and simple, there is a high margin of error in converting SLOC data into function points (Wong et al., 2008). This research is going to bridge this gap by developing a modified backfiring algorithm that converts COCOMO II to Function Point Analysis, converts the result of Function Point to Thousand Lines of code (KLOC) and gives an average software costing from both models.

3. Methodology

Backfiring is the direct mathematical conversion of lines of code data into equivalent function point data. The backfiring equations are bidirectional, they also provide a powerful way of sizing, or predicting source code volume for any known programming language or combination of languages. The availability of empirical data from projects that use both Function Point and lines of code metrics has led to the discovery of the backfiring technique. Table 4 shows conversion ratio of SLOC to FP.

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 497-510
https://doi.org//10.47514/kjcs/2024.1.3.007
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Table 4: Source lines of code per Function Point (Qualitative Software Management, 2023)

Language	SLOC/UFP	Language	SLOC/UFP
ABAP (SAP)	28	Natural	40
ASP	51	.NET	57
Assembler	119	Oracle	37
Brio +	14	PACBASE	35
C	97	Perl	24
C++	50	PL/I	64
C#	54	PL/SQL	37
COBOL	61	Powerbuilder	26
Cognos Impromptu Scripts +	47	REXX	77
Cross System Products (CSP) +	20	Sabretalk	70
Cool:Gen/IEF	32	SAS	38
Datastage	71	Siebel	59
Excel	209	SLOGAN	75
Focus	43	SQL	21
FoxPro	36	VB.NET	52
HTML	34	Visual Basic	42
J2EE	46	First Generation Language	320
Java	53	Second Generation Language	107
JavaScript	47	Third Generation	80
JCL	62	Fourth Generation Language	20
LINC II	29	Fifth Generation	5
Lotus Notes	23		

The major concept behind the backfiring model is the assumption that there is a directly proportional (linear) relationship that can be established for a given programming language. Such a linear relationship is given in equation 3.1 used to convert SLOC to FP.

$$FP = k \times SLOC \quad (3.1)$$

Where SLOC is a count of the logical lines of code in the subject software, and k is a constant based on the programming language.

Similarly, the inverse of the linear relationship can be used to convert FP to SLOC as shown in equation 3.2.

$$SLOC = k' \times FP \quad (3.2)$$

Where k' is an inverse based on the programming language, i.e $\frac{1}{k}$ and FP is function point.

The new model considered Programming Language to be used to develop the system or Language generation, Function Point and Source lines of Code were considered to modify the backfiring algorithm,

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 497-510
https://doi.org//10.47514/kjcs/2024.1.3.007
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

and new variable Complexity adjustment Factor (CAF) is introduced which is the result of general system characteristics, and when the variable CAF is not available in a given problem (1.08) is used as average.

To convert Source Lines of Code to Function Point we used equation 3.3.

$$FP = \frac{SLOC}{k} \times CAF \quad (3.3)$$

Where *SLOC* is Source Lines of Codes in thousand and *k* is the programming language source lines of codes per function point from the programming language table as shown in table 3.1.

The proposed model checks the dataset type KLOC or FP, if the dataset is FP it will convert it to SLOC using backfiring algorithm and pass the result (SLOC) for COCOMO II model computation and find the Estimated Project Cost. The FP will be used for Function Point Analysis and find the Development Cost. Also, If the dataset is KLOC it will convert it to FP using a backfiring algorithm and pass the result (FP) for Function Point Analysis and find Development Cost. The Development Cost of the FPA and Estimated Development Cost of the COCOMO II model will be used together to find the Average Software Development Cost as illustrated in figure 2.

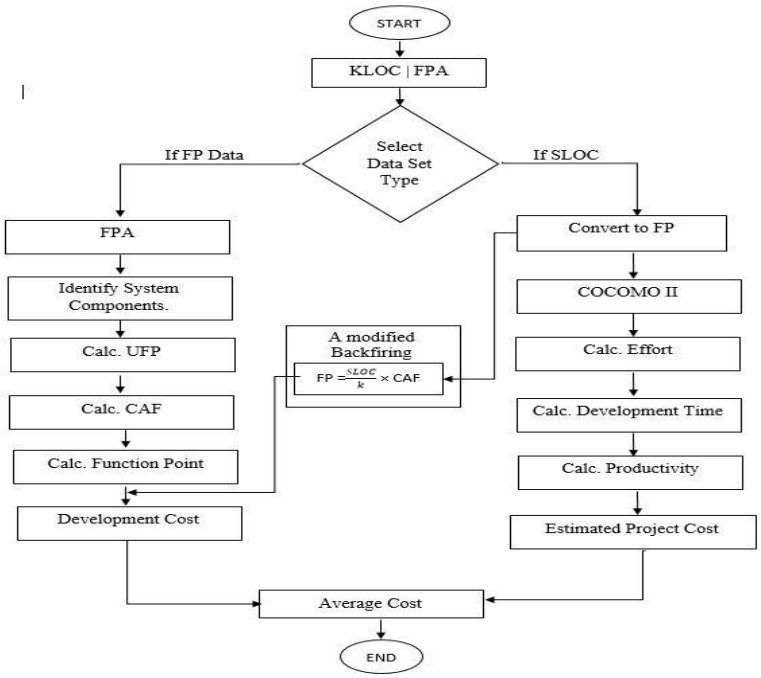

Figure 2: Flowchart of Methodology

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 497-510
https://doi.org//10.47514/kjcs/2024.1.3.007
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

4. Results and Discussion

Figure 3 shows the COCOMO II Analysis interface, the user begins by entering the actual Source Lines of Code (SLOC), which the system will convert to KLOC (SLOC/1000). The user then adjusts the Scale factors using radio buttons and provides the Effort multipliers through similar controls. Additionally, the user selects the programming language for backfiring to perform Function Point Analysis. After specifying these inputs, the system calculates and displays the results for both the COCOMO II and Function Point analysis. The user can also adjust the Complexity Adjustment Factor (CAF) as needed, ensuring a refined and comprehensive analysis.

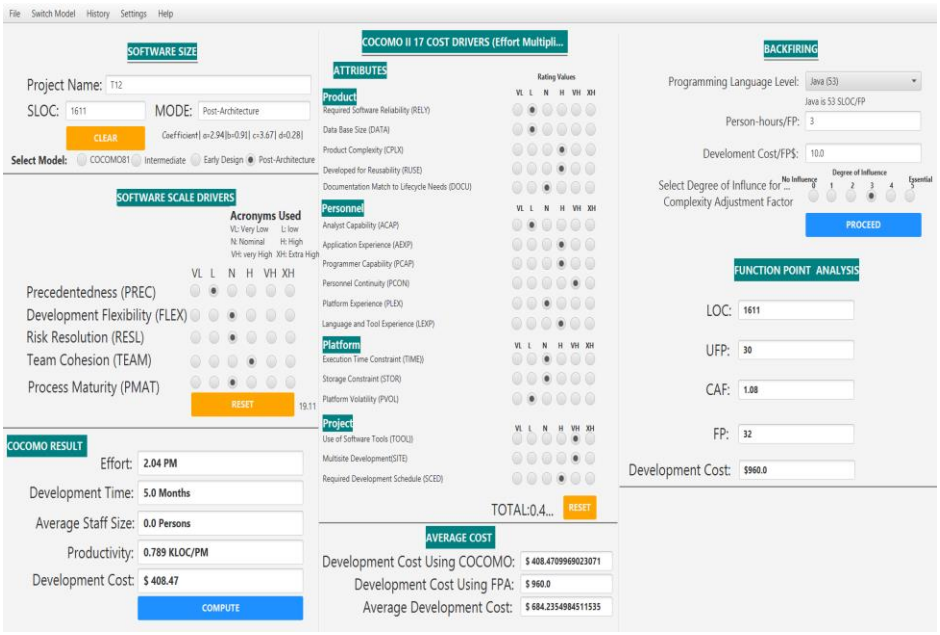

Figure 3: COCOMO II Analysis Interface

The COCOMO II dataset in Lines of Codes was collected from Manalif E. (2013), who obtained his data for testing purposes from the Turkish dataset and the Industry dataset, the backfiring was done using Java programming to obtain Function Point values. The proposed model testing result is shown in Tables 5. Additionally, the NASA dataset is available in the appendix section for further reference and testing.

Kasu Journal of Computer Science Vol. 1 No. 3 [September, 2024], pp. 497-510
https://doi.org//10.47514/kjcs/2024.1.3.007
Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Table 5: COCOMO II Dataset test result (Manalif E. 2013)

Dataset	Size (KLOC)	Actual Effort (PM)	Estimated Effort	Absolute Error (AE)	Proposed						
					Effort (PM)	Absolute Error (AE)	Duration(Months)	Productivity	Backfiring		
									UFP	CAF	AFP
T01	3.00	1.20	7.61	6.41	3.37	2.17	5	0.889	56	1.08	60
T02	2.00	2.00	4.97	2.97	2.86	0.86	5	0.700	37	1.08	39
T03	4.25	4.50	10.97	6.47	7.43	2.93	7	0.572	80	1.08	86
T04	10.00	3.00	26.93	23.93	32.31	29.31	11	0.309	188	1.08	203
T05	15.00	4.00	41.22	37.22	58.26	54.26	13	0.257	283	1.08	305
T06	40.53	22.00	117	95.00	27.73	5.73	10	1.462	764	1.08	710
T07	4.05	2.00	10.42	8.42	2.87	0.87	5	1.411	76	1.08	82
T08	31.854	5.00	90	85.00	147.14	142.14	18	0.216	601	1.08	649
T09	114.28	18.00	605.41	587.41	297.60	279.60	25	0.384	2156	1.80	2328
T10	23.11	4.00	64.88	60.88	64.00	60.00	14	0.361	435	1.08	469
T11	1.37	1.00	3.34	2.34	0.80	0.20	3	1.716	25	1.08	27
T12	1.61	2.10	3	0.90	2.04	0.06	5	0.789	30	1.08	32
Mean Absolute Error (MAE):				**76.41**	**MAE:**	**48.18**					

Figure 4 shows the performance comparison graph between the actual effort, estimated effort, and proposed effort test result from Table 4.1 Turkish dataset in Lines of Codes. The graph demonstrates that the proposed model performs better than the estimated effort model, as it runs below the estimated effort line and is closer to the actual effort line. This indicates that the proposed model provides a more accurate prediction of effort with the available dataset.

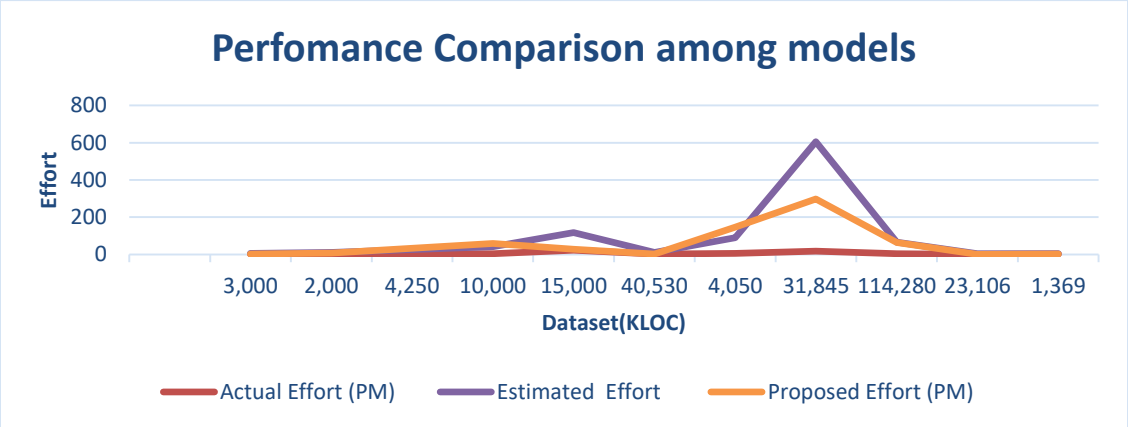

Figure 4: Performance Comparison Graph 1

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Datasets: The study utilizes datasets collected from Lavazza & Robiolo (2010), Manalif E. (2013), and NASA, ensuring a comprehensive evaluation across different sources.

The proposed model outperforms traditional estimation methods, achieving lower Mean Absolute Error (MAE) and Mean Magnitude Relative Error (MMRE). Cross validation confirms the model's robustness and generalizability.

5. Conclusion

This research successfully demonstrates that a hybrid model combining Function Point Analysis and COCOMO II can provide more accurate software effort estimates and improve productivity in software development context. This approach offers a practical solution for project managers, enhancing decision-making and project outcomes in software development. The proposed model achieved an MAE of 48.18 outperforming the estimated model, which had an MAE of 76.81, in comparison 48.18 score is less than 76.81 from the result of the LOC dataset. The improved accuracy of the proposed model has significant implications for software project management, potentially reducing cost overruns and improving project timelines. Limitations include the size and diversity of datasets; which future research could address by incorporating more varied data. This research work indicates directions for further research. Future work should explore the application of the model in different contexts and with larger datasets to validate its generalizability. Compare the Model with Common Software Measurement International Consortium (COSMIC) COSMIC Function Point and COCOMO II Model

References

Ahmed, M., Ibrahim, N. B., Nisar, W., Ahmed, A., Junaid, M., Soriano Flores, E., & Anand, D. (2024). A hybrid model for improving software cost estimation in global software development. *Computers, Materials & Continua, 78*(1), 1399-1422.

Bajusova, D., Silhavy, P., & Silhavy, R. (2024). Enhancing Software Effort Estimation with Self-Organizing Migration Algorithm: A Comparative Analysis of COCOMO Models. *IEEE Access*. 12(1) 67170 -67188

Frank, J. (2019). Enrichment of accurate software effort estimation using fuzzy-based function point analysis in business data analytics. *Neural Computing and Applications*, *31*(5), 1633-1639.

Gupta, R. G., Dumka, A., & Mazumdar, B. D. (2024). Software cost estimation: A comparative analysis. In *2024 International Conference on Computer, Electrical & Communication Engineering (ICCECE)*, IEEE, (Kolkata, India) 1-8

Online ISSN: 1597-2216
Print ISSN: 1597-2178
www.kjcs.edu.ng

Hai, V. V., Thi Kim Nhung, H. L., & Hoc, H. T. (2019). A review of software effort estimation by using functional points analysis. *Proceedings of the Computational Methods in Systems and Software*, (Cham, Switzerland), 408-422.

Jones, C. (1996). Backfiring: Converting lines of code to function points. *Quality Plus Technologies Inc.*, *28*(11), 87-88.

Kaur, M., & Sehra, S. K. (2014). Particle swarm optimization-based effort estimation using Function Point analysis. In *2014 IEEE International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, (Ghaziabad, India), 140-145.

Khan, J. A., Khan, S. U. R., Khan, T. A., & Khan, I. U. R. (2021). An amplified COCOMO-II based cost estimation model in global software development context. *IEEE Access*, *9*, 88602-88620.

Manikavelan, D., & Ponnusamy, R. (2020). Software quality analysis based on cost and error using fuzzy combined COCOMO model. *Journal of Ambient Intelligence and Humanized Computing*, (1), 1-11

Odion, P. O., & Alfa, P. E. (2022). An Optimal Software Development Process Task Sharing Techniques Using Effort Multiplier Value of Software Project Management. *Benin Journal of Advances in Computer Science*, *7*(1), 27-37.

Odion, P. O., & Onibere, E. A. (2014). Modification of Software Constructive Cost Model II (COCOMO II) To Suite Nigerian Development Environment. *Academy Journal of Science and Engineering*, *8*(1), 78-95.

Pauline, M., Aruna, P., & Shadaksharappa, B. (2013). An Enhanced Model to Estimate Effort, Performance and Cost of the Software Projects. *ICTACT Journal of Soft Computing*, *3*(3), 524-533.

Purushotham, J., & Kumar, M. N. (2022). A Study on Software Cost Estimation Models Using Machine Learning. *Journal of Interdisciplinary and Multidisciplinary Research (JIMR)*, 10(17), 47-57.

Putri, R. R., Siahaan, D., & Fatichah, C. (2024). COCOMO II-FG-HGWO: A Modified Hunting Gray Wolf Optimization for Improving Constructive Cost Model II Performance. *International Journal of Intelligent Engineering & Systems*, *17*(4), 932 - 937.

Qualitative Software Management (2023). QSM Resources/ Function Point Languages Table Retrieved September, 2023, from https://www.qsm.com/resources/function-point-languages-table

Rahman, M., Roy, P. P., Ali, M., Gonc¸alves, T., & Sarwar, H. (2023). Software Effort Estimation using Machine Learning Technique. *Journal of Software Engineering Research and Development*, 14(4), 822-827.

Uzzafer, M. (2016). Principle Component Analysis of Function Point Elements. *International Journal of Advanced Science and Technology*, *91*, 39-48.

Wong, J., Ho, D., & Capretz, L. F. (2008). Calibrating function point backfiring conversion ratios using neuro-fuzzy technique. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *16*(6), 847-862.