Software Team Productivity Factor in Constructive Cost Model for Software Development Effort Estimation

Okure U. Obot *okureobot@uniuyo.edu.ng*
Department of Computer Science, Faculty of Science
University of Uyo
Uyo, Nigeria

Edward N. Udo *edwardudol@uniuyo.edu.ng*
Department of Computer Science, Faculty of Science
University of Uyo
Uyo, Nigeria

Peter G. Obike *obike.peter@mouau.edu.ng*
Department of Computer Science, College of Physical and Applied Sciences
Michael Okpara University of Agriculture
Umudike, Nigeria

Abstract

One of the models used to implement software development effort estimates is the Constructive Cost Model (COCOMO) and the attributes of this model are said to contain some level of imprecision. This study was motivated by the need to accurately estimate software development effort and also reduces the imprecision contained in the COCOMO. A neuro-fuzzy constructive cost model by Kaur et al., (2018) was studied and found to contain some of the desirable features of a neuro-fuzzy approach. It handles imprecision using Adaptive Neuro-Fuzzy Inference System (ANFIS) with a large dimension of datasets and does not consider software team members productivity. This work introduces software team productivity factor into the conventional COCOMO and converts it to COCOMO II using model definition manual and Rosetta Stone and also considers reducing the number of inputs from 23 to 6. With data gathered from PROMISE repository (NASA project), an ANFIS-based model was built. The new model with the productivity factor was implemented along with that of Kaur et al., (2018) in the MATLAB 2021 programming environment. Findings reveal that with 6 out of the 23 attributes of PROMISE datasets, the ANFIS model (Hybrid and Back Propagation) with the productivity factor performs better than the Kaur et al., (2018) model. The implication is that the productivity of the team members working on a software project can add up or reduce the actual person-hours (Effort) required to develop a software. During the experiments, six (6) important COCOMO inputs that software managers should place more emphasis on during the planning stage were identified

Keywords: Software Development Effort, COCOMO, Team Productivity Factor, Principal Component Analysis, ANFIS, Back Propagation, Hybrid Learning Algorithm.

1. INTRODUCTION

Software development effort estimation (SDEE) predicts or approximates the development time, cost, required workforce and the most realistic amount of effort (expressed in terms of person-hours or money). SDEE is needed to develop a software product using the available incomplete, uncertain, and noisy input (Carbonera et al., 2020; Chatzipetrou et al., 2015; Shivakumar et al., 2016; Bilgaiyan et al., 2017; Rijwani and Jain, 2016; Rehmana et al., 2021).Incomplete, uncertain and noisy in the above definition means that at the early stage of software development, only a little amount of information is available (Kaushik et al., 2013). Effort estimates computed at the

early stages of the software development lifecycle (SDLC) are typically associated with uncertainty (Sehra et al., 2016).

In software engineering, estimation is one of the critical activities and plays a vital role in the development of software for the establishment of cost assessments and delivery timelines (Bilgaiyan et al., 2017; Marapelli and Peddi, 2020). This estimation affects software cost and required Effort and consequently influences the overall success of software development process (Mustapha, 2018). Estimation is usually used as a basis for resources allocation and aids significantly in decision making and management of complex and large software projects (Humayun and Gang, 2012; Mohsin, 2021). It plays an important role in the design of project plans, budget allocation, investment analysis, and pricing processes (Rehmana et al., 2021).

It is quite problematic and more challenging to derive accurate effort estimates for software projects, especially at the early stages of SDLC, because estimations are predictions and there are some levels of uncertainties in predictions. If the estimation is delayed until the requirement specification phase, the inaccuracy of the estimation can be reduced, therefore giving a more reasonable and accurate estimation (Sehra et al., 2016), but in many situations and in practical scenarios, estimations are needed before the specified requirements are elaborated. Estimations are therefore carried out at the early stages of SDLC, even though uncertainty decreases and knowledge increases regarding the product as the project progresses (Arifin et al., 2017).

Various factors affect SDEE. Usman et al., (2017) highlighted the factors that affect estimation accuracy to include the size of software, the experience of team members and their skills, the number of nonfunctional requirements, the geographical distribution of the team members and the level of communication provided by the customer. Sehra et al., (2016) asserted that factors such as effort requirements uncertainty, software size, and the experience of the estimator, inconsistent and incomplete data, frequent requirements changes and the environment's dependency affect accurate SDEE. Other factors are the developer's experience, complexity and impact of changes to the underlying system (Tanveer, 2017). Additionally, estimation models perform differently in different environments and in different software project types (Sehra et al., 2017).

Considering these multiple factors, estimation of Effort in software development can be very difficult, leading to overestimation or underestimation. Both overestimation and underestimation of software effort may lead to risky consequences and sometimes can cause complete failure of a software project. It is important to note that the software industry is a major contributor to the world's economy. Therefore, inaccurate SDEE will cause a great loss in this regard (Humayun and Gang, 2012).

Overestimation may lead to resource wastage, suboptimal delivery time (Nassif et al., 2019) and resources misallocation, which affects the development of other important projects (Mohsin, 2021). It may cause too many resources to be committed to a project and loss of contract bids for software projects (Gultekin and Kalipsiz, 2020). Conversely, underestimation of software development Effort may cause delay and over-run costs, which may eventually results in software project failure (Moosavi and Bardsiri, 2017). It may also lead to project understaffing, excess budgeting expenses, poor management decisions, building a software system with poor quality and late delivery of software products (Nassif et al., 2019; Gultekin and Kalipsiz, 2020).

In recent years, many approaches for SDEE have been proposed. These approaches or methods are divided into two groups: algorithmic and non- algorithmic. Algorithmic methods are based on mathematical methods which try to calculate the relationship between software product factors and the software development effort (Khazaiepoor et al., 2020). Some of the well-known algorithmic methods are regression models, Planning Poker, Delphi and Constructive Cost Model (COCOMO) (Carbonera et al., 2020; Amazal and Idri, 2014; Nassif et al., 2016). Non- algorithmic methods are based on active analysis of factors of the software project and nonlinear characteristics (Shilhavy et al., 2017). Some of such methods are machine learning models,

Analogy Based Estimations (ABE) and expert judgment (Khazaiepoor et al., 2020). The popular estimation methods used are ABE, expert judgment, function points, software sizing, and Bayesian methods (Bilgaiyan et al., 2016; Soni and Kohli, 2017). It is worth mentioning that no single method is considered the best, but using a combination of some methods increases estimation accuracy (Shekhar and Kumar 2016).

Many authors have used COCOMO datasets to implement software development effort estimates despite the fact that the attributes in the COCOMO-based NASA dataset possess a certain level of imprecision. The attributes are analyst capability, programmer's capability, application experience, modern programming practice, and use of software tools, virtual machine experience, language experience, schedule constraint, main memory constraint, database size, and a time constraint for CPU, turnaround time, machine volatility, process complexity, required software liability and the physical kilo line of source code. For example, it is quite impossible to determine a programmer's capability for the problem domain during the planning stage of software development; it is not also certain at the early stage of SDLC which modern programming practice to employ. Software planning is definitely associated with some uncertainties.

This work introduces software team productivity factor into the COCOMO by adapting the work of Kaur et al., (2018); a Neuro-Fuzzy Constructive Cost Model, which carries some of the desirable features of a neuro-fuzzy approach, such as learning ability and good interpretability, while maintaining the merits of the COCOMO model.

One of the main objectives of software development is to improve productivity for the production of more software while reducing its development cost (Mizuno et al., 2000; Ramirez and Oktaba, 2018). Productivity has a direct relationship with the efficiency and effectiveness of the software development process (Melo et al., 2013), and this may be a single item in the entire software quality process (Morasca and Russo, 2001).

The competition witnessed within the software production industries requires timely delivery of software products in such a way that it creates the need for increased team members' performance that will help the industries to remain in the software business (Canedo and Santos, 2019). Software team productivity is measured in order to reduce software development costs, improve the quality of deliverables, and increase the rate at which software is developed (Sudhakar et al., 2012).

Productivity in the software development environment is described by the relationship between the size of the delivered software (input from the development process) and the Effort expended in building the software (output from the development process). Therefore, the general productivity equation is given by (Aquino et al., 2009; Vasilescu et al., 2015):

$$\text{Productivity} = \text{Size}/\text{Effort}$$

Productivity is the ratio of outputs to consumed resources (Card, 2006). Software productivity includes complexities of both software and people, which can be calculated by dividing software size by the cost of development. Measuring productivity helps in identifying under-utilized resources (Nwelih and Amadin, 2008).

Despite efforts to define productivity, there is no consensus in the software industry regarding what the term productivity means and, instead of having only one metric or factor that describes productivity, it is defined by a set of aspects – People, Product, Organization, and Open Source Software Projects (Mato et al., 2021).

In this work, the software team productivity factor is introduced into the COCOMO II model and an ANFIS model was built, with the data obtained from the PROMISE repository, to estimate software development effort. The model was compared to an existing neuro-fuzzy constructive cost model by Kaur et al. (2018). Section 2 of this work presents the related work, section 3

reviews the constructive cost model by Kaur et al., (2018) and shows the architecture of the proposed system, section 4 shows the effect of PCA, while Section 5 built the ANFIS model (both back propagation and hybrid) with the software team productivity factor. In Section 6, the ANFIS model is trained, and its performance is evaluated and compared with Kaur et al., (2018) model. The work is concluded in Section 7.

2. RELATED WORKS

Several authors have used the COCOMO NASA datasets, with the uncertainties in its attributes, in conjunction with other models to estimate software development effort. Suharjito et al., (2016) used a neuro-fuzzy model optimized with PSO to derive an improved software development effort estimation model using NASA dataset software project. The results of the optimization were trained using ANFIS to get an effort prediction. Rijwani and Jain (2016) proposed the use of artificial neural network-based model using multi-layered feed-forward neural network which was trained with a back propagation method and COCOMO data-set was used to test and train the network. Mustapha (2018) performed data mining on COCOMO NASA datasets using three (3) machine learning techniques: Naïve Bayes, Logistic Regression and Random Forests. The generated models were tested using five-folds cross-validations and were evaluated using Classification Accuracy, Precision, Recall, and AUC. The estimation results were then compared to the COCOMO estimation. Khazaiepoor et al., (2020) used a three-phase hybrid approach (feature selection using genetic algorithm and perceptron neural network, associating impact factor with the selected features using linear regression method and optimization of feature weights using an imperialist competitive algorithm) to estimate software development effort. Three datasets (COCOMO, Maxwell and Albrecht) were used to validate the model. Singal et al., (2020) estimated software development effort using differential evolution algorithms to improve the parameter values for algorithmic models like CoCoMo and CoCoMo II. Sharma and Vijayvargiya (2020) carried out neuro-fuzzy computing through ANFIS using COCOMO and COCOMO II datasets by comparing the expected and the actual data. The results showed that ANFIS model could be efficiently used for estimating software development effort. Marapelli and Peddi (2020) estimated Effort using three machine learning techniques (Naïve Bayes, Logistic Regression and Random Forests) that were applied to a preprocessed COCOMO NASA benchmark data, which covered 93 projects. The generated models were tested using five folds cross-validation and were evaluated using Classification Accuracy, Precision, Recall, and AUC. The estimation results were then compared to COCOMO estimation. Zakaria et al., (2021) used several machine learning algorithms (Linear Regression, Support Vector Machine, Regression Tree, Random Forest) to estimate software development effort and the best learning algorithm compared with the COCOMO. Rehmana et al., (2021) investigated five machine learning techniques (polynomial linear regression, ridge regression, decision trees, support vector regression, and multilayer perceptron) for effort estimation using seven standard datasets (Albretch, Desharnais, COCOMO81, NASA, Kemerer, China, and Kitchenham)

All these researches used the COCOMO model in its conventional state, despite its weakness and uncertainties. Other authors tried to enhance and amplify the COCOMO model by complementing it with another effort calculating model or by improving the model's parameter values. Garg et al., (2014) complemented the COCOMO model using the Function Point Analysis. COCOMO estimated the cost of software development from design to the integration phase, while Function Point estimated the cost for the remaining phases. The enhanced model was tested on 20 projects with defined values of cost drivers. Singal et al., (2020) used a differential evolution approach to improve the parameter values of COCOMO and COCOMO 11 and compared the results with the original COCOMO models. Khan et al., (2021) amplified COCOMO 11 with global software development cost drivers for different estimates in the context of global software development.

Rai et al., (2021) designed a model using the features of Support Vector Regression and COCOMO to predict software effort with the help of software development team size. The results showed that with a team size of 51 – 60, the model yielded an accuracy of 92% in predicting

software effort. This means that the accuracy of the model increases as the team size increases. Increasing the team size without considering other factors will result in overestimation.

This work therefore considers the size of the software team members as well as their productivity by introducing software team productivity factor into COCOMO to help remove the problem of underestimation and overestimation since the productivity factor is majorly responsible for both firing and dropping in effort estimation.

3. REVIEW OF KAUR'S SOFTWARE DEVELOPMENT EFFORT MODEL

The software development effort evaluation model in Kaur et al., (2018) is a Neuro-Fuzzy Constructive Cost Model which carries some of the desirable features of a neuro-fuzzy approach, such as learning ability and good interpretability, while maintaining the merits of the COCOMO model. The model deals effectively with imprecise and uncertain input and enhances the reliability of software cost estimates. In addition, it allows input to have continuous rating values and linguistic values, thus avoiding the problem of similar projects having high different estimated costs.

The inputs to this model are the software size and ratings of 22 cost drivers including 5 scale factors (*SFRi*) and 17 effort multipliers (*EMRi*). The output is the software development effort estimation. Ratings of cost drivers can be continuous numerical values or linguistic terms such as "low", "nominal" and "high". The parameters in this model are calibrated by learning from industry project data. This system covers all-important dimensions of software evaluation through the integration of different technologies. Figure 1 shows the architecture of the Kaur et al., (2018) model.

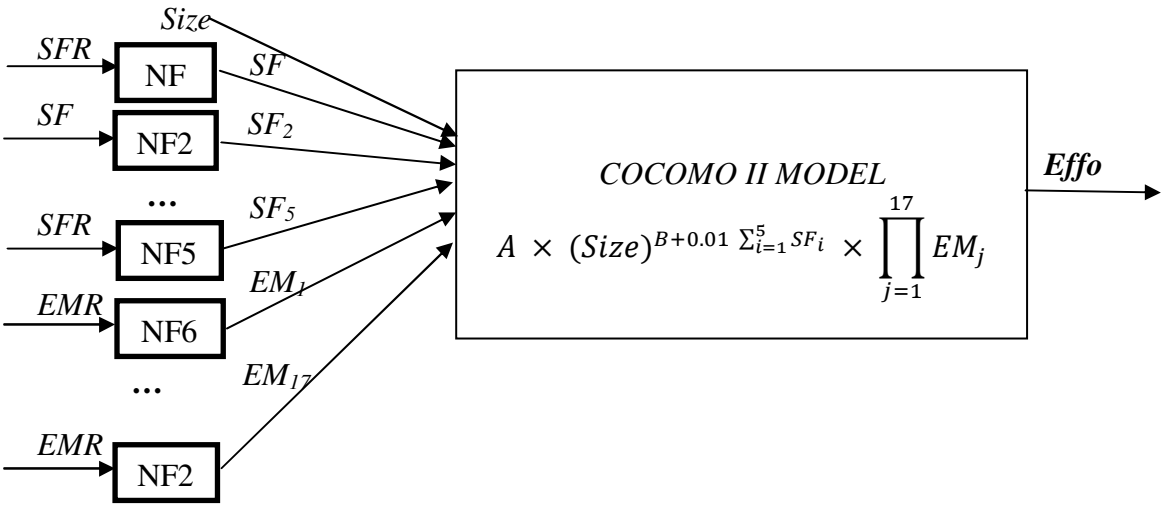

FIGURE 1: Model of the Adapted Model (Kaur et al., 2018).

After careful analysis of the Kaur's model, the following problems are identified:

i. The model only addressed the effort prediction based on existing Neuro-fuzzy model.
ii. The Effort was accurate to an extent because COCOMO II model parameters were not optimized enough
iii. The model considered only 18 projects as data set.
iv. The model took no consideration of the software team members productivity

3.1 Kaur's Model with Software Team Productivity

The productivity coefficient is introduced into the COCOMO model to reduce the effect of productivity among team members. Hanchate and Bichkar (2015) states that the productivity of

an employee or team, σ is given by the ratio scope, ℓ of project and performance η of the employee:

$$\sigma = \frac{Scope\ of\ Project}{Performance} = \frac{\ell}{\eta} \quad (1)$$

The productivity, σ by COCOMO-II is given by the ratio of Effort and Team Size (TS).

$$\sigma = \frac{Effort}{Team\ Size} \quad (2)$$

From (1) and (2), it can be seen that an increase in productivity has a direct proportional increase in the Effort. But the idea is to reduce Effort, hence an inverse of (2) gives:

$$\frac{1}{\sigma} = 1/(\frac{Effort}{Team\ Size}) = \frac{Team\ Size}{Effort} \quad (3)$$

From (3), it is clear that the effort decreases when the right hand side increases. This work termed the right hand side of (3) the productivity constraint. By this, the improved model is derived and depicted in Figure 2

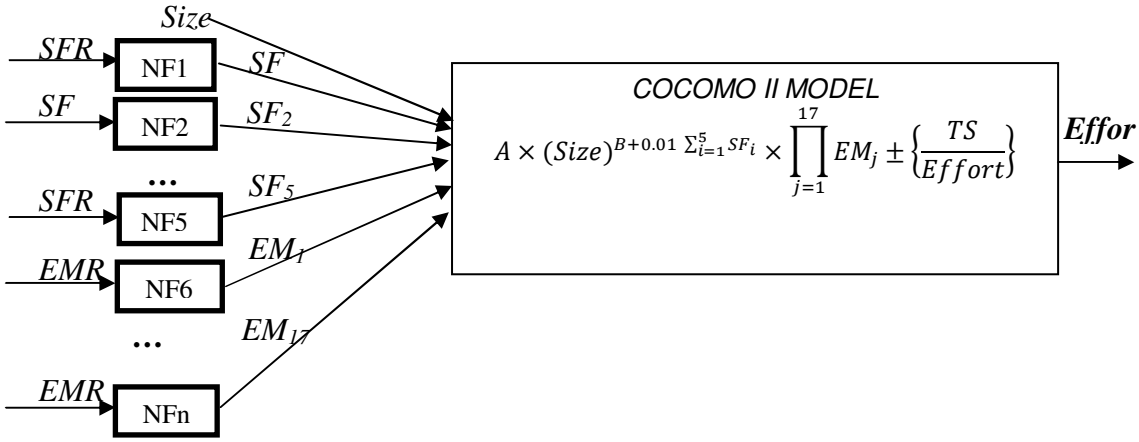

FIGURE 2: COCOMO II Model with Team Size Productivity.

3.2 Architecture of the Proposed System

The proposed system is designed as the integration of components that will enable pre-processing of the dataset, clustering of the preprocessed dataset, creation of an initial Fuzzy Inference System (FIS) from the dataset, as well as subjecting the result to the Neuro-Fuzzy Model to predict the final Effort required to develop a software. The study employs deductive reasoning methodology with pre-specified questions, outcome-oriented data collections, numerical estimation and statistical inference analysis. The architecture of the proposed system is as shown in Figure 3.

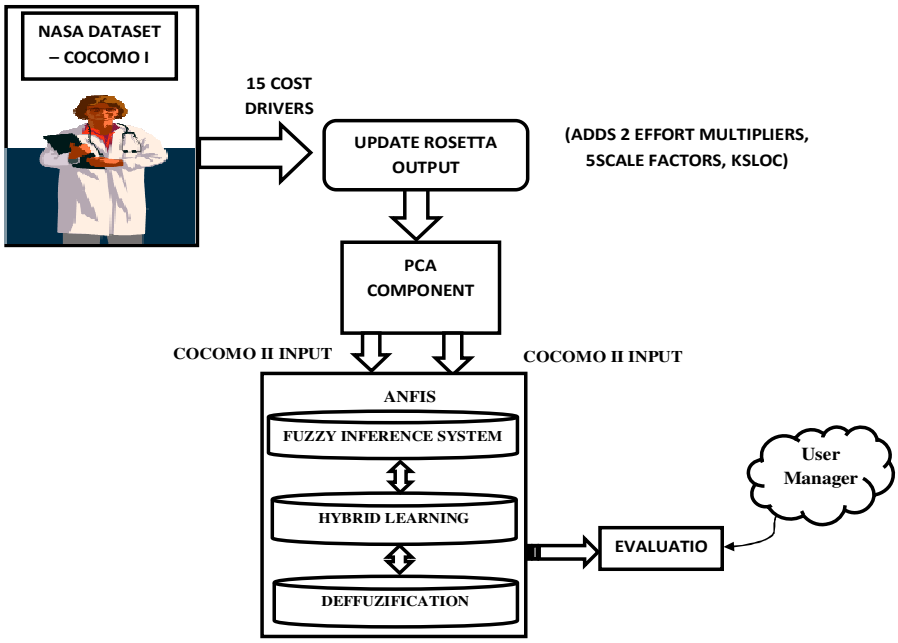

FIGURE 3: Architecture of the Proposed System.

Rosetta stone tool converts original COCOMO 81 files to a form that is compatible with COCOMO II. The following steps were adopted for the conversion:

1. **Update of Size**: Table 1 provides guidelines for converting size in Delivered Source Instruction (DSI) to Source Line of Code (SLOC) for their use with the COCOMO II model.

TABLE 1: Converting Size Estimates.

COCOMO 81	COCOMO II
- DSI	- $SLOC^3$
- 2^{nd} generational Languages	- reduce DSI by 35%
- 3^{rd} generational languages	- reduce DSI by 25%
- 4^{th} generational languages	- reduce DSI by 40%
- object oriented languages	- reduce DSI by 30%

The histogram showing the DSI plot in COCOMO I Dataset and SLOC in COCOMO II Dataset is presented in Figure 4 and 5 respectively.

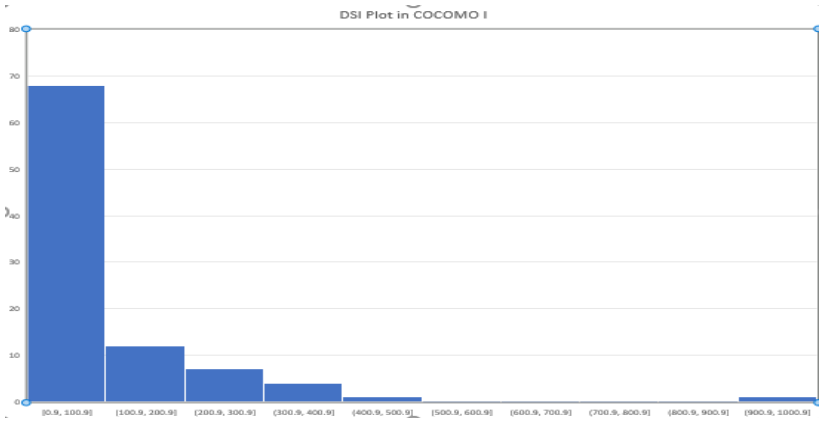

FIGURE 4: DSI Plot in COCOMO I Dataset.

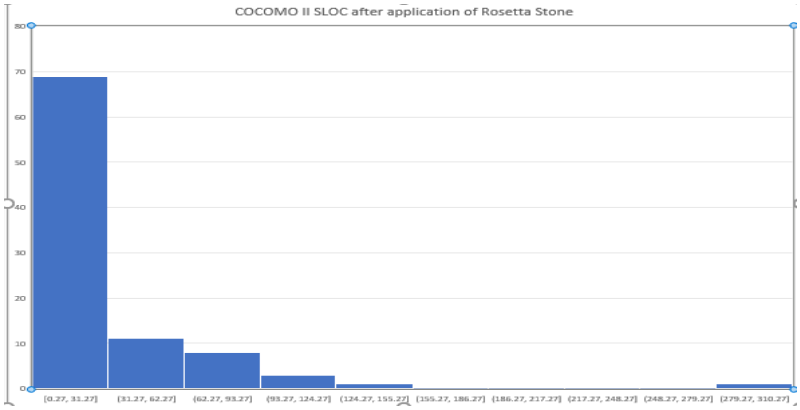

FIGURE 1: SLOC in COCOMO II after conversion.

2. **Conversion of Exponent:** The exponent variables in COCOMO II are five scale factors which are absent in COCOMO I. They are Precedentedness (PREC), Development Flexibility (FLEX), Architecture/Risk Resolution (RESL), Team Cohesion (TEAM), and Process Maturity (PMAT). Table 2 presents values used for the five scale factors that determine the exponent of COCOMO II.

TABLE 2: Mode/Scale Factor Conversion Ratings.

MODE/SCALE FACTORS	ORGANIC	SEMI-DETACHED	EMBEDDED
Precedentedness (PREC)	XH	H	L
Development Flexibility (FLEX)	XH	H	L
Architecture/Risk Resolution (RESL)	XH	H	L
Team Cohesion (TEAM)	XH	VH	N
Process Maturity (PMAT)	MODP	MODP	MODP

PMAT replaces the Modern Programming Practice (MODP) cost driver in the COCOMO I model. A Modern Programming Practices (MODP) cost driver from Table 3 with ratings Very Low (VL) or Low (L) translates into a PMAT rating of VL, or a L level on the SEI CMM scale. A MODP rating of Nominal (N) translates into a PMAT rating of L, or a High (H) Level 1. A MODP rating of H or Very High (VH) translates into a PMAT rating of N or CMM Level 2. As with the other factors, if the project's actual rating is different from the one provided by the Rosetta Stone, the actual value is used. This was achieved by using this formula in Microsoft Excel:

=IF (OR (T2="vl", T2="l"),"vl", IF (OR (T2="h", T2="vh"),"n",IF(T2="n","l")))

Where T2 is the MODP cost driver.

3. **Conversion of Cost Drivers**: The Rosetta Stone guidelines in Table 3 are used to convert COCOMO I cost drivers to COCOMO II.

TABLE 3: Cost Driver Conversion.

COCOMO 81 DRIVERS	COCOMO II DRIVERS	CONVERSION FACTORS
RELY	RELY	Same as actual
DATA	DATA	Same as actual
CPLX	CPLX	Same as actual
TIME	TIME	Same as actual
STOR	STOR	Same as actual
VIRT	PVOL	Same as actual
TURN		Use values of COCOMO Attributes
ACAP	ACAP	Same as actual
PCAP	PCAP	Same as actual
VEXP	PEXP	Same as actual
AEXP	AEXP	Same as actual
LEXP	LTEX	Same as actual
TOOL	TOOL	Use values of COCOMO Attributes
MODP	Adjust PMAT settings	IF MODP is rated VL or L, set PMAT to VL, else if N, set PMAT to L, else if H or VH, set PMAT to N
SCED	SCED	Same as actual
	RUSE	Set to N, or actual if available
	DOCU	If mode = organic, set to L, if semi-detached, set to N, if mode = embedded, set to H
	PCON	Set to N, Same as actual if available
	SITE	Set to H, Same as actual if available

4. PRINCIPAL COMPONENT ANALYSIS (PCA)

In order to eradicate redundant input from the feature vector and sort out the best features that contain the most applicable pieces of information from the given dataset, Principal component analysis was used. In this study, a total of 23 features were extracted in the initial stage. These features were then fed to PCA to extract the most valuable features. The results show that out of the 23 features, 17 features that are effort multipliers have less than 2% explained variance and 6 features that are scalar multipliers have above 2% explained variance. The 6 features have cumulatively 94.1% explained variance. The cumulative explained variance value is presented in Table 4 and the screen plot for the cumulative explained variance is depicted in Figure 6.

TABLE 4: Effect of dimension on the cumulative explained variance.

Feature	Cumulative Explained Variance (%)
RELY	63.3385
DATA	16.4827
CPLX	5.8788
RUSE	3.4049
DOCU	2.7814
TIME	2.23903

STOR	1.60532
PVOL	1.43192
ACAP	0.75292
PCAP	0.60549
PCON	0.56600
AEXP	0.45995
PEXP	0.29948
LTEX	0.15349
TOOL	5.89E-28
SITE	2.52E-33
SCED	1.01E-34
PREC	0.0000
FLEX	0.0000
RESL	0.0000
TEAM	0.0000
PMAT	0.0000
SLOC	0.0000

From Table 4, it is observed that the higher the number of dimensions the higher the cumulative explained variance. The total dimension in Table 4 is twenty-three (23), which is the total number of dimensions in the original dataset. To select relevant features from the original dataset, a dimension is chosen in order to achieve 94.1% of the original information retained. In the case of this system, the best minimum number of dimensions is Feature 6 (TIME) with a cumulative explained variance of 2.24%.

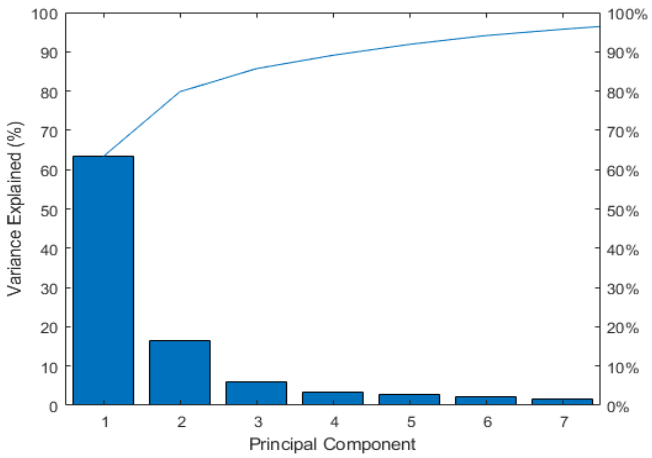

FIGURE 6: Cumulative Screen Plot of the original dataset.

From the results shown in Figure 6, the first six principal components (RELY, DATA, CPLX, RUSE, DOCU, and TIME) have been selected as the new feature vector because these six principal components have a total cumulative variance of 94.1. The algorithm of PCA calculates these scores by making use of covariance matrix, eigenvector, and eigenvalues from the original feature vector. The number of dimensions used was six (6) because six (6) is the minimum dimension that explains more than 94.1% variance in the original dataset. Hence the new (or reduced) dataset had a dimension of six (6).

5. ADAPTIVE NEURO-FUZZY INFERENCE SYSTEM (ANFIS)

ANFIS model is trained using the reduced feature vector generated by principal component analysis. The structure of the ANFIS model shown in Figure 7 comprises the crisp input (represented by the input linguistic variables), Gaussian input membership functions with

parameters; standard deviation,σ, and mean $c$rules, output membership functions, and the crisp output.

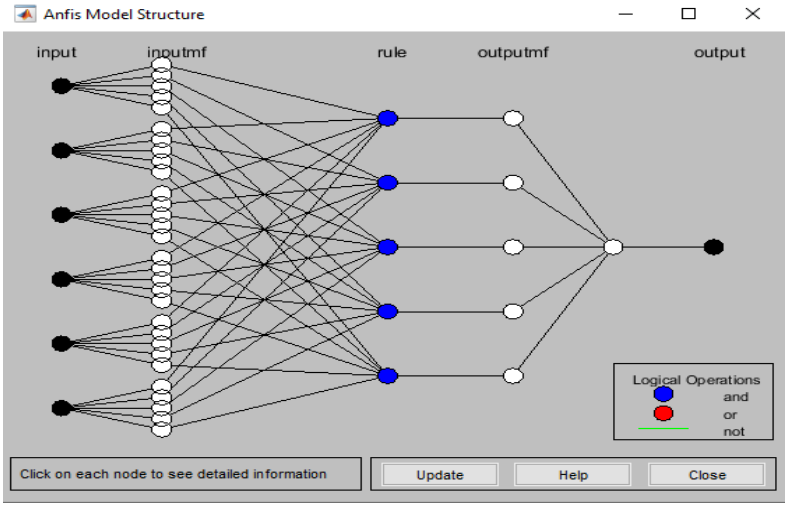

FIGURE 7: The ANFIS structure.

The six inputs (principal components) are fuzzified at ANFIS layer 1. Each input variable is made up of five membership functions (very low, low, normal, high, and very high). The ANFIS model successfully learned from the data and produced an optimal Fuzzy Inference System (FIS). This initial FIS generated using Subtractive Clustering is composed of a rule base with 5 rules for 5 clusters, and the membership function used is the Gaussian membership functions, and auto-generated rule-base. The number of rules generated by the ANFIS model is 5 computed with Subtractive Clustering with n input, n rules and n mf using Cluster Influence Range of 0.5.However, the value of the cluster Influence Range was set to 0.55 which reduced the rules generated to 5 and data points are used as the candidates for cluster centers. For the comparison of Kaur et al., (2018) Model, the number of rules generated when using Grid Partitioning is 64 (sixty-four) computed as the number of linguistic terms (T) to the power of the number of linguistic variables (V) - T^V.

The components of the FIS generated by the ANFIS model is presented in Figure 8; the optimized FIS structure. The membership function plots for RELY, DATA, CPLX, RUSE, DOCU and TIME are respectively shown in Figures 9 – 14.

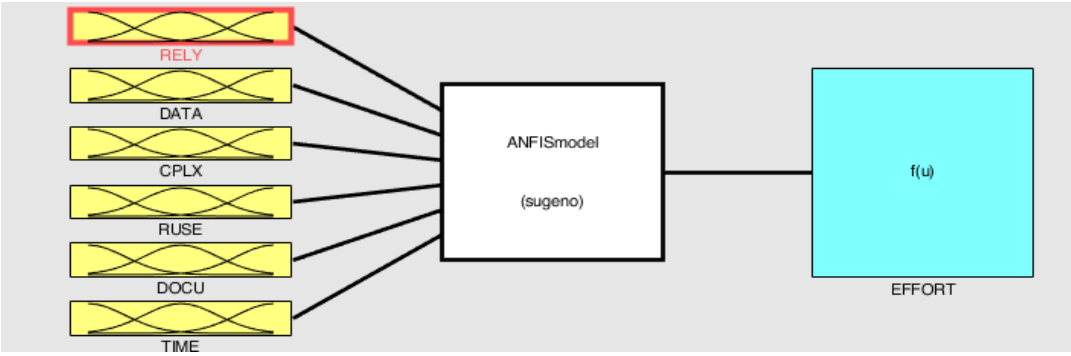

FIGURE 8: Optimized FIS structure.

From Figure 9 to Figure 14, the ranges of each input to 4 decimal places are: RELY [-89.2013,889.9036], DATA [-2.0196,3.9218], CPLX [-4.5516, 2.6066], RUSE [-0.5891, 1.8484], DOCU [-0.9441, 0.6666], and TIME [-0.4301, 0.5891].

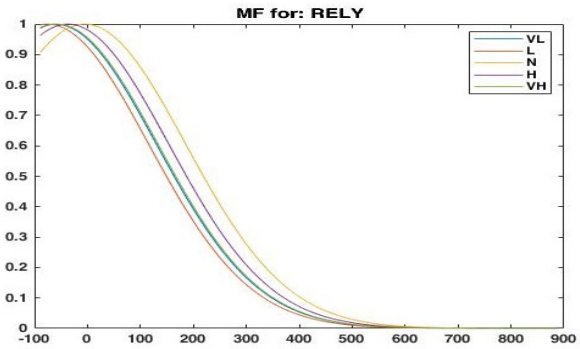

FIGURE 9: PC1-RELY membership function.

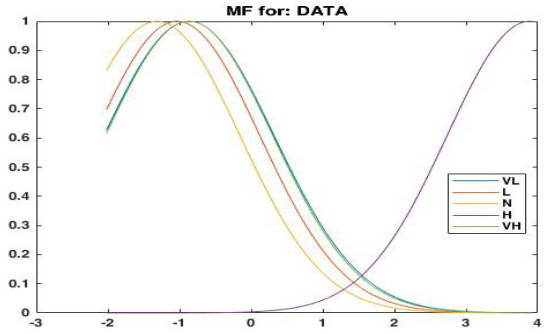

FIGURE 10: PC2-DATA membership function.

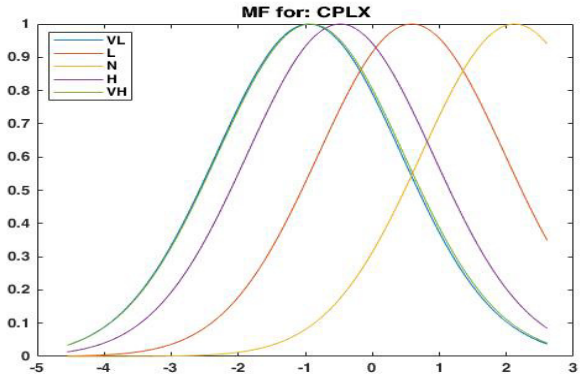

FIGURE 11: PC3-CPLX membership function.

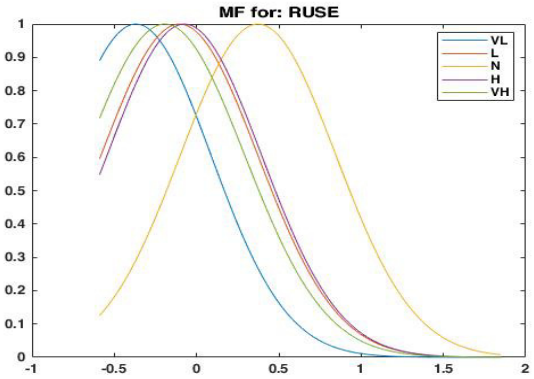

FIGURE 12: PC4-RUSE membership function.

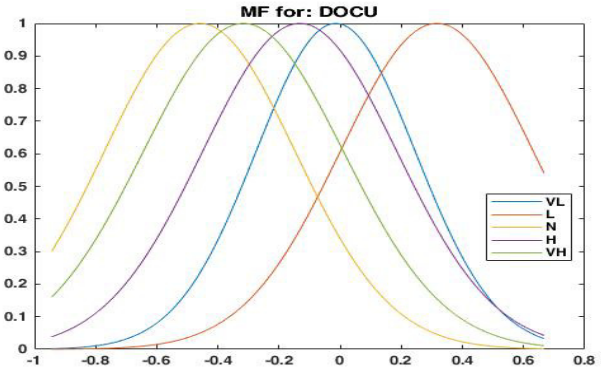

FIGURE 13: PC5-DOCU membership function.

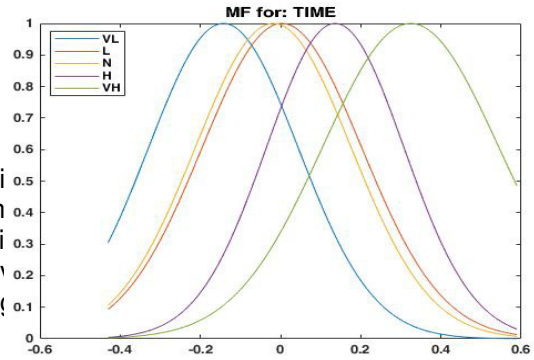

FIGURE 14: PC6-TIME membership function.

There are 64 rules generated using Grid Partitioning while 5 rules are generated for 5 clusters using the subtractive clustering tool. The rules are used by the fuzzy inference engine to reason on the given input value in order to produce a crisp output. The five (5) rules generated for the inference system are given in Figure 15.

1. If (RELY is in1cluster1) and (DATA is in2cluster1) and (CPLX is in3cluster1) and (RUSE is in4cluster1) and (DOCU is in5cluster1) and (TIME is in6cluster1) then (EFFORT is out1cluster1) (1)
2. If (RELY is in1cluster2) and (DATA is in2cluster2) and (CPLX is in3cluster2) and (RUSE is in4cluster2) and (DOCU is in5cluster2) and (TIME is in6cluster2) then (EFFORT is out1cluster2) (1)
3. If (RELY is in1cluster3) and (DATA is in2cluster3) and (CPLX is in3cluster3) and (RUSE is in4cluster3) and (DOCU is in5cluster3) and (TIME is in6cluster3) then (EFFORT is out1cluster3) (1)
4. If (RELY is in1cluster4) and (DATA is in2cluster4) and (CPLX is in3cluster4) and (RUSE is in4cluster4) and (DOCU is in5cluster4) and (TIME is in6cluster4) then (EFFORT is out1cluster4) (1)
5. If (RELY is in1cluster5) and (DATA is in2cluster5) and (CPLX is in3cluster5) and (RUSE is in4cluster5) and (DOCU is in5cluster5) and (TIME is in6cluster5) then (EFFORT is out1cluster5) (1)

FIGURE 15: Auto-generated rule-base.

The resulting relationship among the ANFIS input variables can be visualized using the surface plots presented in Figures 16 to 25.

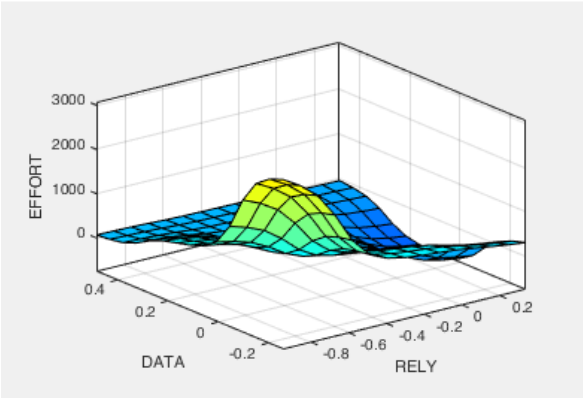

FIGURE 16: Effect of PC1-RELY on PC2-DATA.

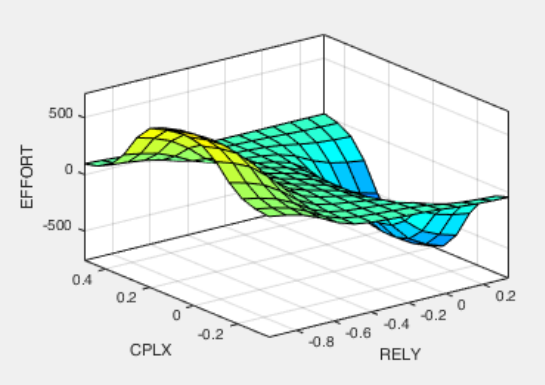

FIGURE 17: Effect of PC1-RELY on PC3-CPLX.

In Figure 16, Effort is fixed and non-increasing if there is no database to implement and if the reliability requirement of the system is minimal. The plot shows that a high software reliability requirement and database size increases the number of persons required to develop the software. In Figure 17, the Effort is high as the process complexity of the program to be developed increases. It can also be depicted that the reliability requirement has more effect on the Effort because the process complexity gets low as the reliability requirement (RELY) decreases.

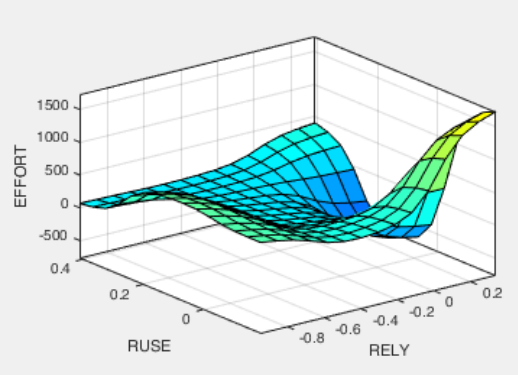

FIGURE 18: Effect of PC1-RELY on PC4-RUSE.

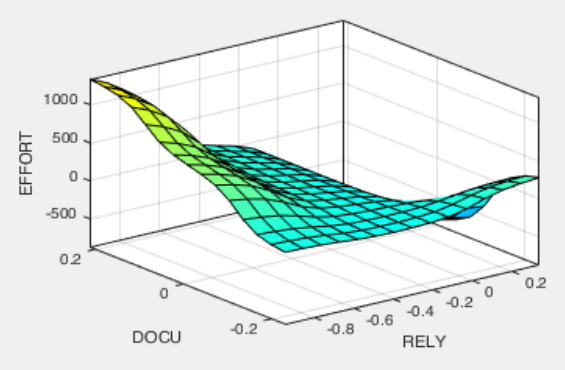

FIGURE 19: Effect of PC1-RELY on PC4-DOCU.

In Figure 18, when the required reusability feature of software is high and the required reliability is low, it affects the number of effort required to develop the software. Also, an increasing reusability constraint can also increase the reliability requirement and number of Effort required to develop such software. In Figure 19, the Documentation match to life-cycle (DOCU) can increase to a high extent the number of Effort because the more the documentation the more functions and processes complexity of the system. However, even though detailed documentations contribute to a good software but when the required reliability constraint is low, the number of Effort reduces. It can also be seen from the plot that the effort remains fixed as the required reliability constraint is below or equal to 0 (not constrained).

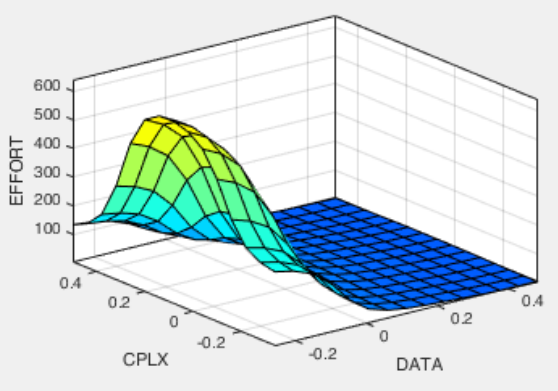

FIGURE 20: Effect of PC2-DATA on PC3-CPLX.

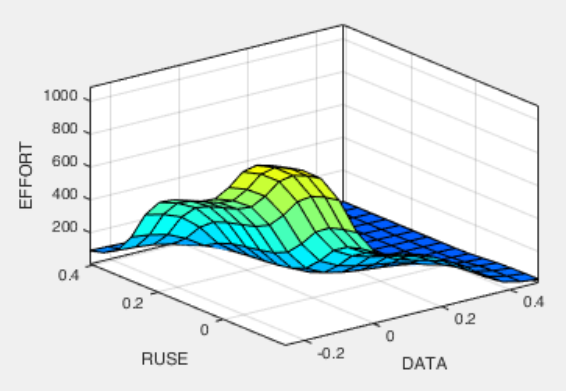

FIGURE 21: Effect of PC2- DATA on PC4-RUSE.

In Figure 20, it can be seen that the Effort required to develop software increases when the process complexity of the software and database size (DATA) are the required constraints but as the process complexity constraint reduces and database size reduces, the Effort required is fixed to 120 and not zero. The 120-person-month minimal Effort in the plot accounts for the time and man power required to create the table and schema of the database regardless of the complexity of the system.

In Figure 21, as the required reusability reduces and the database size increases, the number of person-month (Effort) required to develop a software increases from 100 person-months to 395 person-month but when the database size becomes significant for large projects for example, and the reusability constraint is 0.2 – 0.4, effort increases to 600 person-month.

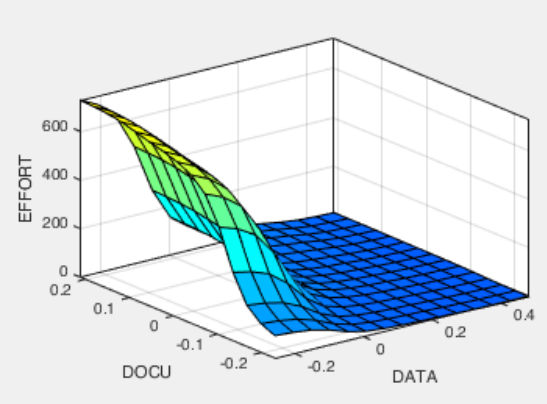

FIGURE 22: Effect of PC2- DATA on PC5-DOCU.

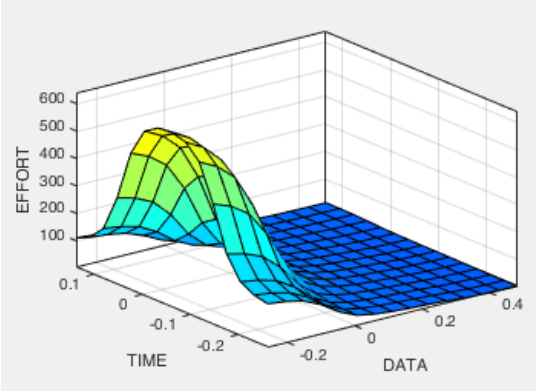

FIGURE 23: Effect of PC2- DATA on PC6-TIME.

In Figure 22, the number of persons per month required to develop software is 600-persons-month when Documentation match to life-cycle (DOCU) is detailed and the database size is large at 0.4. Even when the database size is large and the Documentation match to life-cycle (DOCU) small (less than zero), the number of person-month required to develop the software is minimal. In Figure 23, it can be seen that the person-month required developing software increases as the execution time (TIME) and database size (DATA) of the software are required constraints. However, as the execution time constraint reduces and database size reduces, the effort decreases to a non-zero minimum (120 person-month) and not zero. The 120 person-month minimal Effort in the surface accounts for the man-hours required to create the table and schema of the database.

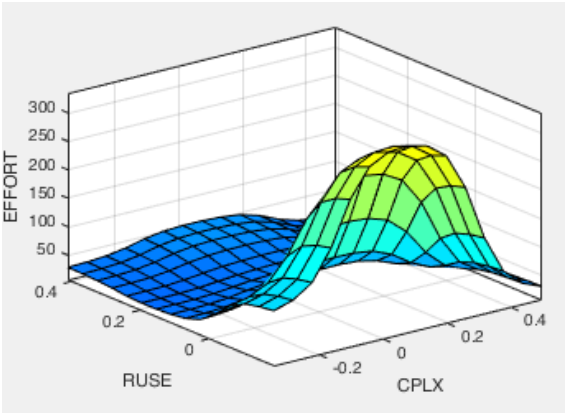

FIGURE 24: Effect of PC3- CPLX on PC4-RUSE.

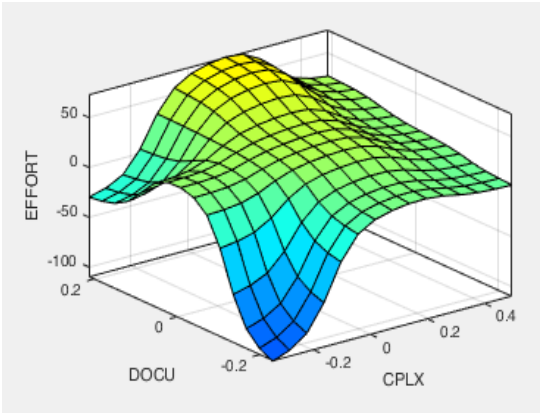

FIGURE 25: Effect of PC3- CPLX on PC5-DOCU.

Figure 24 shows that the person-hours required to develop software increase as the process complexity increases. The effect of the required reusability constraint compared to the process complexity is infinitesimal. Also, when the complexity constraint reaches the peak value of 0.4 and the reusability requirement constraint is negative, the Effort remains at a minimum value of 30-person-hours. Figure 25 shows the effect of process complexity (CPLX) and Documentation match to life-cycle (DOCU) on the Effort. The surface shows that more detailed documentation increases the Effort and complexity.

6. PERFORMANCE EVALUATION

The model was trained with 70% of the entire dataset and tested with 30% of the dataset. The model training was carried out on different epoch for Kaur et al., (2018), ANFIS back propagation

(BP) and hybrid models (HB) for purposes of comparison. The effect of the model testing is shown in Table 5.

TABLE 5: Comparison of Predictions and COCOMO.

SNO	COCOMO Model	Kaur Model	HB ANFIS Model	Rel. DIFF	BP ANFIS Model
1	2	1.63	2.02	0.07%	-1929.282
2	2	1.63	2.02	0.07%	-1929.282
3	2	1.63	2.02	0.07%	-1929.282
4	8	3.20	0.33	7.45%	99.827
5	8	1.04	6.02	0.67%	98.329
...	...	...	...	...	...
19	48	147.34	61.54	3.57%	58.849
20	50	161.38	42.71	1.03%	111.816
21	60	2063.10	59.22	0.01%	118.441
22	60	2217.61	48.31	2.15%	130.590
23	60	700.23	90.62	13.46%	-142.057
24	60	320.10	85.82	9.8%	162.348
25	60	270.02	66.44	0.67%	327.549
...	...	...	...	...	...
30	72	2461.35	19.67	29.61%	-54.742
31	82	453.63	81.31	0.01%	427.410
32	90	374.97	119.13	8.18%	160.002
33	97	45243.13	191.20	59.26%	378.720
...	...	...	...	...	...
91	4178	45243.13	4179.06	0%	4834.174
92	4560	271398.02	4558.77	0%	5099.861
93	8211	64507.11	8210.53	0%	6608.818

The Rel. DIFF column in Table 5is the Relative Difference between the COCOMO Effort and ANFIS effort. It shows HB ANFIS prediction with less than a 1% difference from the COCOMO effort, relative to other predictions; it indicates software projects with overestimated productivity of team size (as in SNO. 5, 21, 25, 31, 91, 92 and 93) and underestimated productivity of team size (as in SNO. 1, 2, and 3).

Rel. DIFF is calculated using this statistical model:

$$100\left(\frac{|x-xref|}{|xref|}\right)\left(1-e^{\frac{-|x-xref|}{a}}\right)\% \quad (4)$$

Based on experiments conducted in sections 4 and 5, it can be seen that ANFIS model (with the productivity factor) using Hybrid Training Model with six (6) inputs reduced by PCA gave better estimate than the Kaur et al., (2018) Model, and the Back Propagation Model with six (6) inputs reduced by PCA. To further investigate this, two performance metrics (MMRE and MSE) were used to compare the results of Kaur et al., (2018) model and the ANFIS models. Table 6 shows the performance evaluation result.

TABLE 6: Performance Evaluation Result.

Measure	Kaur 2018 Model	BP ANFIS Model	HB ANFIS Model
MMRE	100.4684	388.2579	31.4829
MSE	1.8300	0.000082	0.00000187

From the result, it is deduced that the hybrid training model of ANFIS is more efficient and stable in terms of reduced error during training. The MRE Performance Measure of Kaur et al., (2018) Model, Hybrid and Back Propagation were plotted. The plots are depicted in Figures 26 to 28.

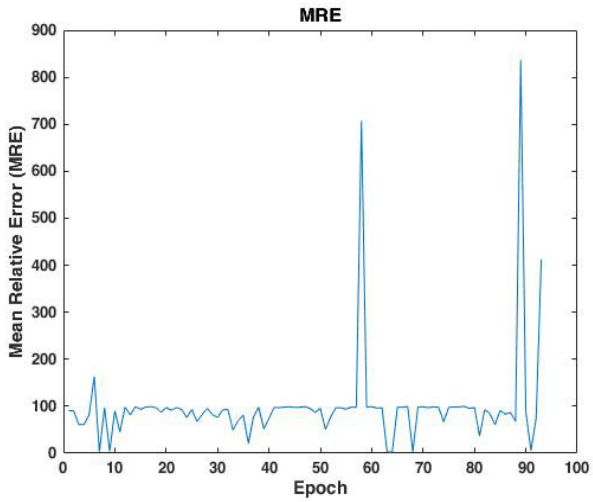

FIGURE 26: MRE Plot for Kaur 2018.

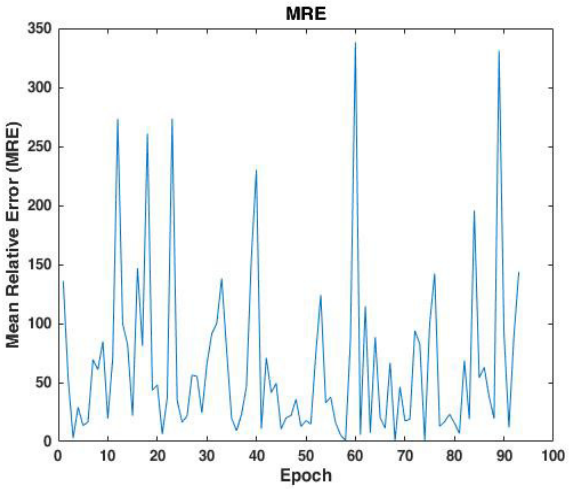

FIGURE 27: MRE Plot for ANFIS BP.

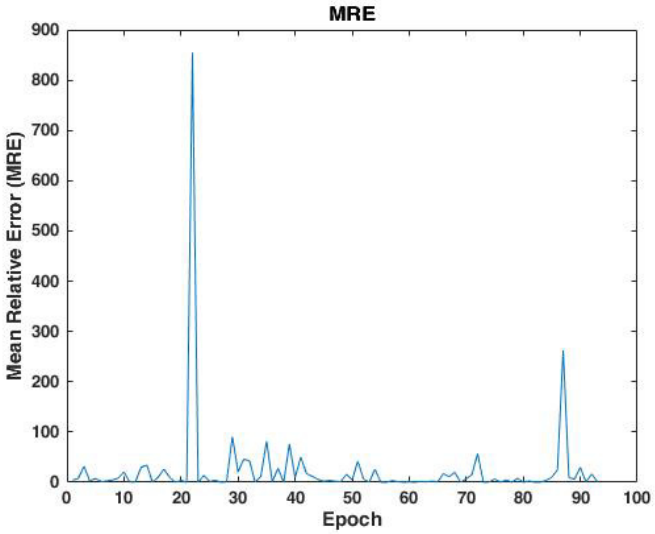

FIGURE 28: MRE Plot ANFIS HB.

The plots showed that the Hybrid training guaranteed convergence more than the other plots. Figure 26 showed that there was a premature convergence from Epoch 10 to Epoch 40, afterward a sudden divergence in Epoch 60 and Epoch 90. This divergence is responsible for some outrageous predictions in Kaur model column in Table 5 compared to the actual. In Figure 27, the high shoots between Epoch 10, 20, 60 and 90 were responsible for negative predictions in BP ANFIS Model column. This is where the least square method complements the back

propagation algorithm to prevent large learning rate that may have occurred at early epochs. In Figure 28, the least square method complements the large learning rate that occurred between epoch 10 and 20. At Epoch 20, the large learning rate was reduced by least-square hence ensuring convergence at Epoch 70.

7. CONCLUSION

This work focuses on SDEE by comparing the work of Kaur at al., (2018) - which handled precision using ANFIS without emphasis on dimension reduction of inputs needed for the estimation, with ANFIS algorithms (back propagation and hybrid) by reducing the number of inputs from 23 to 6 using PCA and also introducing team productivity factor to the existing COCOMO using Hanchate and Bichkar (2015) productivity ratio.

The result shows that ANFIS Model (with productivity factor) using the hybrid training algorithm with 6 inputs performed better than the Kaur et al., (2018) Model and other training algorithm of ANFIS. Also, minor variations in the ANFIS output using the productivity ratio showed how the productivity of team members was underestimated or overestimated.

The ANFIS result using Hybrid learning best predicted the actual Effort in the COCOMO NASA dataset when compared to that of back propagation and Kaur et al., (2018) Model. However, in the course of the work, we found out that productivity of the development team accounts for why there are deviations from the actual effort in the original COCOMO. The productivity of the team members working on a software project can add up or reduce the actual person-hours (Effort) required developing software. For instance, the large number of programmers without requisite knowledge of the project tools and architectural framework will increase the number of hours to deliver a software job, thereby leading to overestimation. Therefore, the productivity of team members is a major factor responsible for both fire and drop in person hours.

The implication is that the effect of fuzzy attributes in estimating the development effort of software professionals and managers can lead to inaccurate effort estimates which in effect can lead to loss-inducing bids, project management problems and poor satisfaction by clients. Estimates that show quality attributes with productivity of software development team members in view can help in estimating near to exact Effort needed to develop a software. The study has achieved the following: software team productivity factor is incorporated into the conventional COCOMO model; the number of inputs has been reduced from 23 to 6 using PCA; better performance in terms of accuracy, precision etc. is obtained than what is seen in Kaur (2018).This will assist software managers and developers in optimizing the performance of their staff and the cost of developing software.

Software productivity involves a wide range of measures. The most critical question is what measure(s) to use. As a way of further research, an attempt should be made to compare the results of several productivity measures as well as explore other areas of productivity other than team size, which is quite easy to count.

8. REFERENCES

Amazal, F. and Idri, A. (2014). Software Development Effort Estimation using Classical and Fuzzy Analogy: A Cross-validation Comparative Study. International Journal of Computational Intelligence and Applications, 13(3), 1450013, doi: 10.1142/S1469026814500138.

Arifin, H. H., Daengdej, J. and Khanh, N. T. (2017). An empirical study of effort-size and effort-time in expert-based estimations, Proceedings of 8th IEEE InternationalWorkshop on Empirical Software Engineering in Practice, (IWESEP 2017), Tokyo, Japan, pp. 35–40. https://doi.org/10.1109/IWESEP.2017.21.

Aquino, G.S., Meira, S.R. (2009). Towards Effective Productivity Measurement in Software Projects. In Proceedings of the Fourth International Conference on Software Engineering Advances (ICSEA), Porto, Portugal, 241–249.

Bilgaiyan, S., Mishra, S. and Das, M. (2016). A Review of Software Cost Estimation in Agile Software Development using Soft Computing Techniques. 2nd IEEE International Conference on Computational Intelligence and Networks (CINE), Bhubaneswar, India, 112–117. https://doi.org/10.1109/CINE.2016.27.

Bilgaiyan, S., Sagnika, S., Mishra, S., and Das, M. (2017). A Systematic Review on Software Cost Estimation in Agile Software Development. Journal of Engineering Science and Technology Review, 10(4), 51–64. https://doi.org/10.25103/jestr.104.08.

Canedo, E. D.; Santos, G.A. (2019). Factors Affecting Software Development Productivity: An Empirical Study. In Proceedings of the XXXIII Brazilian Symposium on Software Engineering (SBES 2019), New York, NY, USA, Association for Computing Machinery, pp. 307–316.

Carbonera, C. E., Farias, K. and Bischoff, V. (2020). Software Development Effort Estimation: ASystematic Mapping Study. Institution of Engineering and Technology Software, 14(4), 328-344.

Card, D. N. (2006). The Challenge of Productivity Measurement. In Proceedings of 24th Annual Pacific Northwest Software Quality Conference, Portland Convention Center, Portland, Oregon, 181 -190.

Chatzipetrou, P., Papatheocharous, E., Angelis, L. and Andreou, A. (2015). A Multivariate Statistical Framework for the Analysis of Software Effort Phase Distribution, Information and Software Technology, Elsevier, **59**, 149–169.

Garg, K., Kaur, P., Kapoor. S. and Narula, S. (2014). Enhancement in COCOMO Model Using Function Point Analysis to Increase Effort Estimation. International Journal of Computer Science and Mobile Computing, 3(6), 565 – 572.

Gultekin, M. and Kalipsiz, O. (2020). Story Point-Based Effort Estimation Model with Machine Learning Techniques, International Journal of Software Engineering and Knowledge Engineering, 30(1), 43–66.

Hanchate D. B. and Bichkar R. S. (2015), Mathematical Modeling of Lewis and COCOMO-II software cost estimation using regulatory focus theory, Applied Discrete Mathematics and Heuristic Algorithms, International Scientific Journal 1(2), 5–25.

Humayun, M. and Gang, C. (2012). Estimating Effort in Global Software Development Projects using Machine Learning Techniques. International Journal of Information and Education Technology, 2(3), 208 – 211.

Kaur I., Narula G., Wason, R., Jain V. and Baliyan A. (2018). Neuro Fuzzy - COCOMO II model for Software Cost Estimation. International Journal of Information Technology, 10(2), 181–187.

Kaushik A., Soni A. and Rachna S. (2013). A Simple Neural Network Approach to Software Cost Estimation, Global Journal of Computer Science and Technology, 13(1), 22 – 30.

Khan, J., Khan, S., Khan, T. and Khan, I. (2021). An Amplified COCOMO-II Based Cost Estimation Model in Global Software Development Context. IEEE Access, 9, 88602 – 88620. doi: 10.1109/ACCESS.2021.3089870.

Khazaiepoor, M., Bardsirs, A. and Keynia, F. (2020). A Hybrid Approach for Software Development Effort Estimation Using Neural Networks, Genetic Algorithm, Multiple Linear Regression and Imperialist Competitive Algorithm. International Journal of Nonlinear Analysis and Application 11(1), 207 - 224.

Marapelli, B. and Peddi, P. (2020). Effort Estimation Methods in Software Development using Machine Learning Algorithms, Parishodh Journal. Vol. IX, Issue 1, 824 – 829.

Melo, C.; Cruzes, D.S.; Kon, F.; Conradi, R. (2013). Interpretative Case Studies on Agile Team Productivity and Management. Information and Software Technology, 55, 412–427. https://doi.org/10.1016/j.infsof.2012.09.004.

Mizuno, O.; Kikuno, T.; Inagaki, K.; Takagi, Y.; Sakamoto, K. (2000). Statistical Analysis of Deviation of Actual Cost from Estimated Cost using Actual Project Data, Information and Software Technology, 42, 465–473.

Mohsin, R. Z. (2021). Application of Artificial Neural Networks in Prediction of Software Development Effort, Turkish Journal of Computer and Mathematics, 12(14), 4186 – 4202.

Moosavi, S. H. S., and Bardsiri, V. K. (2017). Satin Bowerbird Optimizer: A New Optimization Algorithm to Optimize ANFIS for Software Development Effort Estimation, Engineering Applications of Artificial Intelligence, 60, 1-15.

Morasca, S.; Russo, G. (2001). An Empirical Study of Software Productivity. In Proceedings of the 25th International Computer Software and Applications Conference (COMPSAC 2001), Invigorating Software Development, Chicago, IL, USA, 8–12 October 2001; pp. 317–322.

Mota, J.S.; Tives, H.A.; Canedo, E.D. (2021). Tool for Measuring Productivity in Software Development Teams, Information, 12, 396, https://doi.org/10.3390/info12100396.

Mustapha, A. (2018). Predicting Software Effort Estimation Using Machine Learning Techniques, In Proceedings of 8th International Conference on Computer Science and Information Technology, Amman, 249- 256.

Nassif, A. B., Azzeh, M., Capretz, L. F. and Ho, D. (2016). Neural Network Models for Software Development Effort Estimation: A Comparative Study. Neural Computing and Applications, 27(8), 2369-2381, DOI: 10.1007/s00521-015-2127-1.

Nassif, A. B., Azzeh, M., Idri, A. and Abran, A. (2019). Software Development Effort Estimation Using Regression Fuzzy Models. Hindawi Computational Intelligence and Neuroscience, Volume 2019, Article ID 8367214, https://doi.org/10.1155/2019/8367214.

Nwelih, E. and Amadin, I.F. (2008). Modeling Software Reuse in Traditional Productivity Model, Asian Journal of Information Technology, 7(11), 484-488.

Rai, P., Verma, D. and Kumar, S. (2021). A Hybrid Model for Prediction of Software Effort Based on Team Size. IET Software, Wiley, 15, 365 – 375. doi:10.1049/sfw2.12048.

Ramirez-Mora, S.L.; Oktaba, H. Team Maturity in Agile Software Development: The Impact on Productivity. In Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, 23–29 September 2018; 732–736.

Rehmana, I., Alib, Z. and Jana, Z (2021). An Empirical Analysis on Software Development Efforts Estimation in Machine Learning Perspective, Advances in Distributed Computing and Artificial Intelligence Journal, 10(3), 227- 240.

Rijwani P. and Jain, S. (2016). Enhanced Software Effort Estimation using Multi-Layered Feed Forward Artificial Neural Network Technique, In Proceedings of Twelfth International Multi-Conference on Information Processing, Procedia Computer Science, 89, 307-312.

Sehra, S. K., Brar, B. Y. and Kaur, N. (2016). Predominant factors influencing software effort estimation. International Journal of Computer Science and Information Security, 14(7), 107–110.

Sehra, S. K., Brar, Y. S., Kaur, N. and Sehra, S. S. (2017). Research Patterns and Trends in Software Effort Estimation. Information and Software Technology, 91, 1–21. https://doi.org/10.1016/j.infsof.2017.06.002.

Sharma, S. and Vijayvargiya, S. (2020). Enhancing Software Project Effort Estimation using Neuro-Fuzzy System. Solid State Technology, 63(6), 2986 – 2998.

Shekhar, S., and Kumar, U. (2016). Review of Various Software Cost Estimation Techniques. International Journal of Computer Applications, 141(11), 31–34. Retrieved from http://www.ijcaonline.org.

Shivakumar, N., Balaji, N. and Ananthakumar, K (2016): A Neuro Fuzzy Algorithm to Compute Software Effort Estimation, Global Journal of Computer Science and Technology: C - Software and Data Engineering, 16(1), 23 – 28.

Singal, P., Kumari, A. and Sharma, P. (2020). Estimation of Software Development Effort: A Differential Evolution Approach. In Proceedings of International Conference on Computational Intelligence and Data Science, Precedia Computer Science, 167, pp. 2643 – 2652.

Silhavy, R., Silhavy, P. and Prokopova, Z. (2017). Analysis and selection of a regression model for the use case points method using a stepwise approach, Journal of Systems and Software, vol. 125, 1–14.

Soni, D. and Kohli, P. J. (2017). Cost Estimation Model for Web Applications Using Agile Software Development Methodology. Pertanika Journal of Science and Technology, 25(3), 931–938.

Sudhakar, G., Farooq, A. and Patnaik, S. (2012). Measuring Productivity of Software Development Teams, Serbian Journal of Management, 7(1), 65 – 75.DOI: 10.5937/sjm1201065S.

Suharjito, S., Nanda, S. and Soewito, B. (2016). Modeling Software Effort Estimation Using Hybrid PSO- ANFIS, In Proceedings of 2016 International Seminar on Intelligent Technology and Its Application, Lombok, Indonesia, 219 – 224.

Tanveer, B. (2017). Guidelines for Utilizing Change Impact Analysis when Estimating Effort in Agile Software Development. Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE 2017), Kariskrona, Sweden, pp. 252–257. https://doi.org/10.1145/3084226.3084284.

Usman, M., Borstler, J. and Petersen, K. (2017). An Effort Estimation Taxonomy for Agile Software Development. International Journal of Software Engineering and Knowledge Engineering, 27(04), 641–67. https://doi.org/10.1142/S0218194017500243

Vasilescu, B.; Yu, Y.; Wang, H.; Devanbu, P.T.; Filkov, V. (2015). Quality and productivity outcomes relating to continuous integration in GitHub. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, 30 August–4 September 2015, pp. 805–816.

Zakaria, N., Ismail, A., Ali, A., Khalid, N. and Abidiu, N. (2021). Software Project Estimation with Machine Learning. International Journal of Advanced Computer Science and Applications, 12(6), 726 – 734.