Estimation Model for Enhanced Predictive Object Point Metric in OO Software Size Estimation Using Deep Learning

Vijay Yadav
Department of Computer Science and Engineering, Dr A. P. J Abdul Kalam Technical University, India
vijayyadavuiet@gmail.com

Raghuraj Singh
Department of Computer Science and Engineering, Harcourt Butler Technical University, India
raghurajsingh@rediffmail.com

Vibhash Yadav
Department of Information Technology, Dr A.P.J Abdul Kalam Technical University, India
vibhashds10@gmail.com

Abstract: *The Software industry's rapid growth contributes to the need for new technologies. PRICE software system uses Predictive Object Point (POP) as a size measure to estimate Effort and cost. A refined POP metric value for object-oriented software written in Java can be calculated using the Automated POP Analysis tool. This research used 25 open-source Java projects. The refined POP metric improves the drawbacks of the PRICE system and gives a more accurate size measure of software. This paper uses refined POP metrics with curve-fitting neural networks and multi-layer perceptron neural network-based deep learning to estimate the software development effort. Results show that this approach gives an effort estimate closer to the actual Effort obtained through Constructive Cost Estimation Model (COCOMO) estimation models and thus validates refined POP as a better size measure of object-oriented software than POP. Therefore we consider the MLP approach to help construct the metric for the scale of the Object-Oriented (OO) model system.*

Keywords: *Effort estimation, functional size measurement, object orientation predictive object point, software metrics, software measurement.*

Received June 6, 2020; accepted August 29, 2022
https://doi.org/10.34028/iajit/20/3/1

1. Introduction

Software size is critical to the process of the software development process. During the implementation of a software project, the estimated size of the software is used in various ways for effective project planning and control purposes. It is essential to calculate the development process's efficiency after the project has ended. Software size is also a necessary factor for estimating software effort and cost.

Estimating software size before the actual development is a very challenging task. Over the years, various approaches have been proposed to estimate the software size. However, no metrics seem to be the "silver bullet" that addresses all drawbacks and fits all commitments.

Typically, software size metrics are utilized under a specific development environment and paradigm. A range of contending software size estimation approaches used for cost and effort estimation are available in the literature.

This conception is contradictory to the Object-Oriented (OO) model. OO methods distinguish data and processes, while OO designs associate them. Here are several extents to that an OO metric requires accurately calculating effort or efficiency monitoring. Calculating the amount of underdone methodology for software delivery is essential.

Some methods like Function Point Analysis (FPA), original regression, and the Constructive Cost Estimation Model (COCOMO) model are unsuccessful because they are unsuitable for all software developed using different development paradigms. Thus, software size estimation models that give outstanding results for one type of software may not be appropriate for other software applications. For example, FPA Developed for traditional software and gave excellent results could not be extended for the OO software.

Traditional software size, cost, and effort estimation techniques vigorously failed in the case of new software development paradigms. Metrics like Source Lines of Code (SLOC) and FPA were developed in a period that involved programming to divide the resolution space into data and processes.

This concept contradicts the OO model, which distinguishes between the data and processes and associates them with OO designs, Minkiewicz [24]. Thus, good estimation models for OO software are required to provide accurate metrics for effort or efficiency monitoring.

Wittig and Finnic [29] used the Artificial Neural Network (ANN) to estimate the Effort needed for traditional software and found that it produces consistent results. Khoshgoftaar *et al*. [22] presented the ANN procedure for calculating software quality.

Kanmani *et al*. [19] used ANN to predict the Effort for OO applications based on the class points.

In this paper, we Estimate the Effort for OO software through the Refined POP metrics and compare it with actual Effort.

We improved the Effort for OO software by applying Deep Learning. Two different types of Deep -learning methods are designed with training models. The estimation of Effort by CFNN and MLP shows very high accuracy. CFNN and MLP with Bayesian Regularisation (BR) demonstrated a correlation of 98.95 % and 99.85 %, respectively. Figure 1 briefly details various metrics used to measure the software's size.

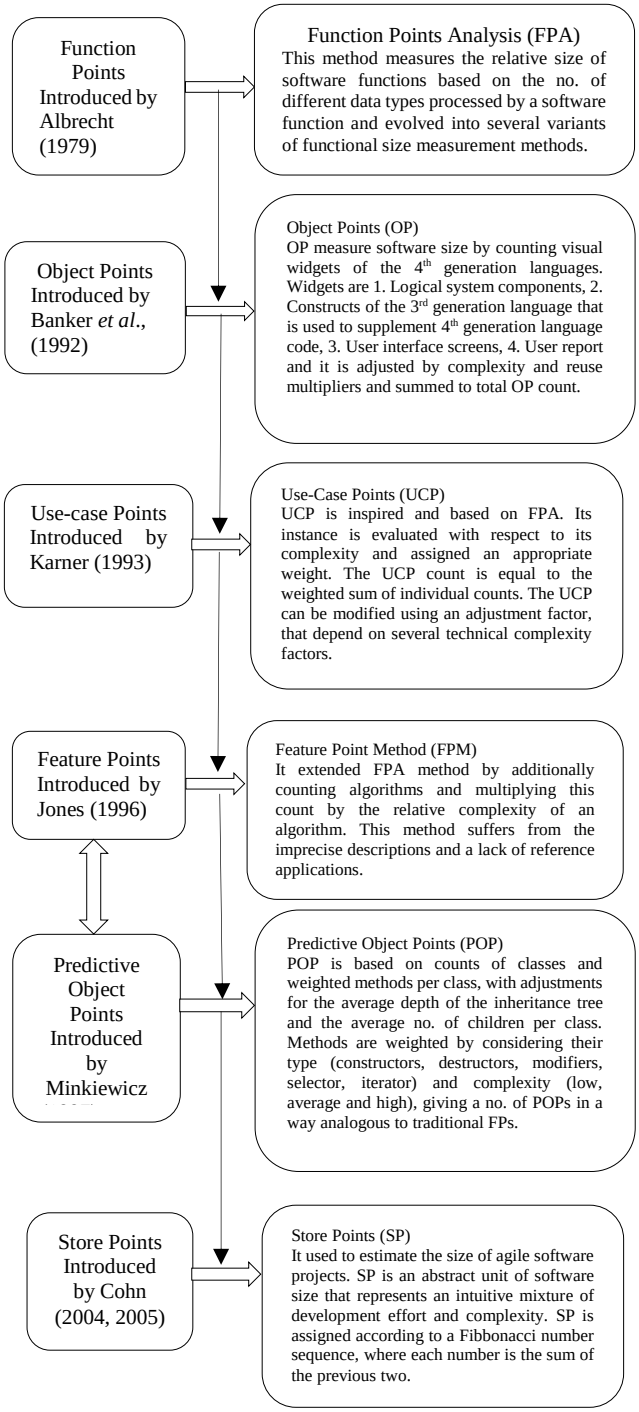

Figure 1. Measuring the size of software.

2. Literature Survey

Enough research on the size, cost, and effort estimation for OO software has been done over the last two decades. There are two major research approaches related to predicting software development effort: developing the size metrics that can be utilized for predicting Effort and developing predictive techniques and models for effort estimation. Lines of Code (LOC) and feature points are the most commonly used size metrics for estimating software effort. Maybe LOC is one of the oldest, Albrecht and Gaffney [5]. Several LOC variants, including Non-Commented lines of code (NCLOC) or Non-Blank Lines Of code (NBLOC), Efficient Lines of code (ELOC), and the number of executable declarations, are introduced and used in industry Fenton and Bieman [10].

FPA is a tool for calculating the size and efficiency of the software. Metric role points assess projects from the end-user standpoint by calculating end-user characteristics, Albrecht [4]. Based on that, FPA represents the software viewpoint of the customers and managers, while LOC represents the view of the developers, Jørgensen [18]. The Java code reviewer tool JCQR was also proposed. It was developed as a free IDE, offering more dynamic options to developers. It is based on Java code standards and can check any piece of Java code saved as a text file. It uses five categories of code rules, Abdallah and Alrifaee [1].

Some of the optimization algorithms performed for software effort estimation, such as the firefly algorithm, Ghatasheh *et al*. [12], Regression fuzzy model, Nassif *et al*., [26], Machine Learning for software fault prediction for OO software metrics Singh *et al*., [28].

PRICE systems have developed POP, and the metric POP has been created to prevent efforts needed to build an OO software system, Minkiewicz [24]. It is based on the Feature Point (FP) system process counting. POPs are intended to boost their use in procedural systems over the FP by building well-recognized metrics in combination with OO systems, Haug *et al*. [15].

POP is reflected to be a more reliable display of OO size than FP, Judge and Williams [20]; Jain *et al*. [17] and can also be used to measure the Effort that can be made to assess the program list and the expense of project software. POPs are a metric for approximating the dimensions of OO applications. The increasing class offers the system high-level inputs that determine the system's organization based on behavior. Nonetheless, there is no reliable POP mapping with the software size. Various metrics are presented in the review section for assembling the superiority of OO software design, such as Aggarwal *et al*. [2] address specifications and present a new set of OO software metrics. Two proposed parameters are calculated, the amount of generosity contained in the project, and then systematically calculated alongside the set of 9 axioms by Weyuker. Chidamber and Kemerer [6, 7] address these

requirements by designing and instigating a new set of OO interface steps.

Few empirical research is presented on the impact of object-oriented metrics on software development quality, such as Chidamber *et al*. [8]. ML is a subfield of Artificial Intelligence (AI) that has been applied in the area Zhang and Tsai [32]; Regolin *et al*. [27]. Several metrics, including effort estimation, have been enhanced using deep learning. A rigorous survey of the software development effort based on Machine Learning (ML) has been done by Akinsanya *et al*. [3]. This work is intended to provide a point of entry for professionals looking to add machine learning to their toolkits for developing software. It classifies recent literature, detects trends, and points out its shortcomings. According to the survey, some writers admit that ML for SD industrial applications has not been as standard as the evidence showed. The ML approach automatically induces information from historical project data in forms such as models, functions, laws, and patterns. Regolin *et al*. [27] have shown how to predict lines of code from (FP) or several components (NOCs) using two ML with Genetic Programming (GP) and ANN, Verner and Tate [30]. The ML algorithms, such as GP and neural networks, are challenged to understand and implement, Idri *et al*. [16].

3. Proposed Effort Estimation Approach

LOC is the most common metric for size estimation, but it is not easy to estimate the size of software expressed in LOC before the actual software development. Also, LOC is a size measure that is dependent on the programming language in which the software is written. Here, we propose an effort estimation approach using refined POP and deep learning through Curve fitting Neural Network (CFNN) and MLP. Here is a list of some popular software estimation methods and their tool in Table 1.

Table 1. Estimation methods with tools comparisons.

METHODOLOGIES	TOOLS
Predictive Object Points (POPs) – measured by metrics: 1) TLC:- top-level classes; 2) DIT:- average depth in inheritance tree; 3) WMC:- weighted methods per class; and 4) NOC – an average of children	PRICE Systems' True S
Object Metric – projects scope as defined in Unified Modeling Language (UML): classes, sub-model components, use cases, and interfaces, capable by size complexity, genericity, as well as reuse	TASSC: Estimator
Use Case Points – According to the use of case diagrams and actors;	Duvessa's Estimate Easy Use Case (EEUC)
Application Points (COCOMO II Level 1) - Used OP that counts reports, screens, and 3GL modules, according to weights complexity	Duvessa's Estimate Easy Use Case (EEUC)
Model-Based Sizing – Arrangement of OO metrics: Use cases number and complexity, classes and objects	QSM's SLIM Estimate

3.1. Predictive Object Points (POP) Metric

POP introduced by PRICE, Minkiewicz [24], Minkiewicz and Fad [25], has established the POP metric and incorporates several standard measures in the review work to determine the appropriate for predicting metric the Effort required to create an OO software system. According to the logic, it's associated with the following 4 OO metrics to describe the metric POPs by Equation (1) Minkiewicz, and Fad [25]; Haug *et al*. [15]; Jain *et al*. [17]:

$$\text{POPs} = \text{AvgWMC} \times \text{TLC} \times \frac{\{1+[(1+\text{AvgNOC})\times\text{AvgDIT}]^{1.01}+(|\text{AvgNOC}-\text{AvgDIT}|)^{0.01}\}}{7.8} \quad (1)$$

The above equation combines the AvgWMC, TLC, AvgNOC, and AvgDIT. These parameters have their own identity. The role of the settings is explained below, Littlefair [23].

1. TLC: It is the total number of classes rooted in a class diagram. These classes are connected at the top level without any parents. All other classes are called the derived class.
2. AvgDIT: It indicates the average number of Depth of Inheritance tree (DIT) values for every class presented in the class diagram. The DIT is a length of inheritance hierarchy trail for a class in which the class originated from the root.
3. AvgNOC: It indicates the average number of NOC values for every class in a class diagram. NOC is the number of classes which is straight inherited from the class.
4. AvgWMC: It indicates the average number of a weighted method for every class, where every type of arrangement is weighted based on complexity.

The weight of a method calculates the AvgWMC metric by its category and complexity. The plans are listed as five method types:

1. Constructor
2. Destructor
3. Modifiers
4. Selector, And an Iterator.

The constructors and destructors are collected into one form because their complexity is similar, Minkiewicz [24]. The methods complications in the class are further divided into low, medium, and high. The types of method weightings with complexity and complexity rules are set by inspecting the amount of actual Effort connected with 100 C++ /Java project methods. Expert information is also extended during this cycle.

In general, however, AvgWMC is unavailable at the design stage from the UML class diagram, Minkiewicz, and Fad [25]; Haug *et al*. [15]. In this respect, we cannot use POPs in the early development process to develop a SLOC prediction model. To tackle this problem, Minkiewicz proposes that AvgWMC should be

determined based on the standard percentage distributions of methods and complex forms.

AvgWMC is generally unavailable at the design stage from the UML class diagram, Minkiewicz, and Fad [25]; Haug *et al*. [15]. For more details." Therefore, POP cannot be used in the early development process to develop a SLOC prediction model. In particular, the following steps can be used for calculating AvgWMC in Figure 2.

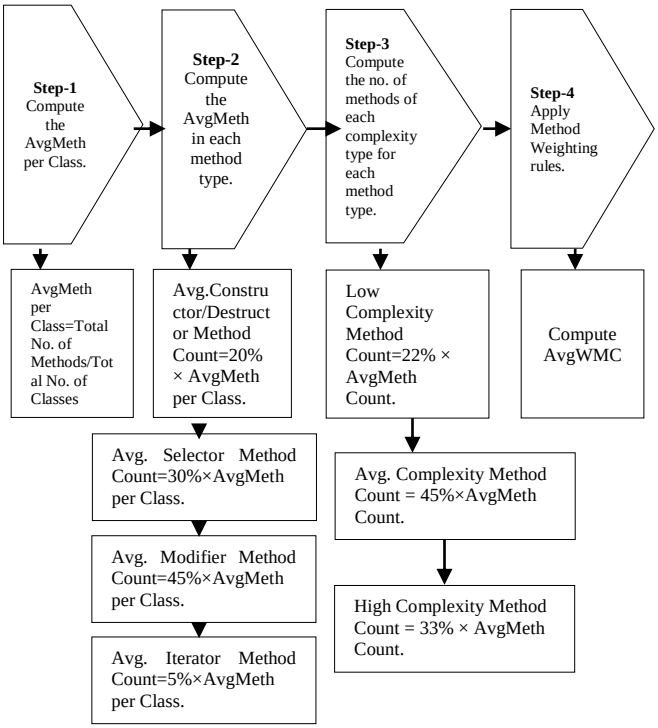

Figure 2. AvgWMC calculation steps.

3.2. Refined Predictive Object Points (POP) Metric

In the true OO environment, as in java projects, the level of reusability through inheritance is always considered to be high, and hence $(|\text{AvgNOC} - \text{AvgDIT}|)^{0.01}$ of Equations (1) can be omitted while estimating Java projects, Jain *et al*., [17]. Thus, the resultant simplified formula for the refined POP is given in Equation (2).

$$\text{REFINED\_POPs} = \text{AvgWMC} \times \text{TLC} \times \frac{\{1+[(1+\text{AvgNOC})\times\text{AvgDIT}]^{1.01}\}}{7.8} \quad (2)$$

The APA automation tool can measure the refined POP count by fragmenting the complete project into each java file and aggregating values as per Equation (2). Besides, simplification was suggested and validated in the POP count formula by implementing a modified object-oriented metric Average Weighted Method Count (AWC) that can be utilized to replace the WMC metric Yadav *et al*. [31], as shown in Equation (3).

$$\text{WMC} = \text{AMC x } 10.478 \quad (3)$$

The authors also recognized the relationship between the original and refined POP counts. This refinement and relationship are valid only for Java-based software and may not be applicable in any other development environment.

3.3. Software Actual Effort Estimation

When the latest technologies grow and the size of the software is immense, the Judgment calculation does not accurately foresee. The need for formula-based evaluation approaches is therefore illustrated. The Constructive Cost Model (COCOMO-I) is a mathematics-based estimation model. In COCOMO-I, the most straightforward calculation is used for Effort in Equation (4) to estimate the number of Person-Months needed for project growth. The measurement effort sequence is described in Figure 3.

$$\text{Effort} = \text{A} * (\text{Size})^{\text{B}} \quad (4)$$

Here *A* is constant in proportionality, and *B* is the economy. Table 2 applies to the values of *A* and *B*, depending on the project category. A project is divided into three kinds:

1. Organic-projects, including small teams with good knowledge
2. Semi-detached projects that have medium-sized organizations with diverse expertise
3. Embedded projects that are built under a set of restrictions.

Table 2. Shows the A and B values for the COCOMO model.

Software/Project	A	B
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

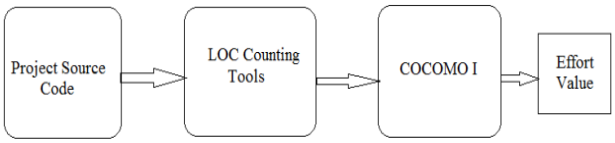

Figure 3. The procedure of effort estimation.

At this point, we summarise the size in terms of actual Effort, KLOC, and estimation through refined POP metric. Since the size of both the projects considered for the study is 2-50 KLOC, we felt them as the organic type and used the formula below for actual effort calculation in Equation (5):

$$\text{Actual Effort} = 2.4 * (\text{KLOC})^{1.05} \quad (5)$$

4. Effort Estimation using Refined POP Metric and Deep Learning

ANN is a computer processing method inspired by an indigenous neuron network. The Feed-Forward Neural Network (FFNN) architecture can be divided into two types: a single layer with adjustable weight and bias and a multi-layer network that includes input layers, a hidden neuron layer, and a neural output layer. The input layer includes self-determining variables interconnected to the hidden layer. The hidden layer

includes activation functions and measures the weights of the variables to investigate the effects of predictors on the (dependent) target variables, Haykin [13]. An ANN consists of three layers: input, hidden layer, and output layers, which means that a network of three layers is shown in Figure 4.

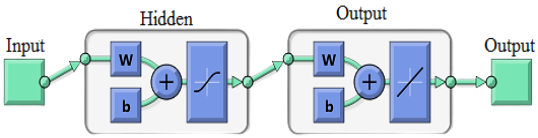

Figure 4. The architectures of Neural Networks.

An MLP-based neural network contains neuron layers with one input/output layer and more than one hidden layer that are optionally accessible. The MLP is created on neurons that calculate the non-linear input vector function and weights of the scalar variables. Neurons in the same layer interpret a signal concurrently and pass it from the input to the output layer. In CFNN, the input and weight vector determine the activation of a hidden neuron. CFNN with enough hidden neurons can arbitrarily implement any finite input-output function. The neural networks for OO application development effort estimation using MLP are generally multi-layered FFNN. The input value in the CFNN is the POP ratio, and the network output is the optimized value of POP.

This research uses CFNN and MLP to estimate Effort using POP and advanced POP metrics. The network input is 1xN, with a 1-dimensional array. The hidden layer size is 25, and two different training algorithms employ a back-propagation training feature. Software development effort is a function of one variable, i.e., software size expressed in POP and refined POP. The enhanced Levenberg-Marquardt (LM), Finschi [11], Heiat [14], and Bayesian regularisation (BR) algorithms have been used to train the network and achieve lower mean squared errors. De Barcelos Tronto *et al*. [9]; Heiat [14]. The LM algorithm is quicker than the BR back-propagation algorithm. Along with the LM and BR back propagation training techniques, we use Mean Square Error (MSE) and determination coefficient to produce results suited for calculating software size, Khalid *et al*., [21]. The complete implementation has been done using MATLAB inbuilt function library and code.

5. Result Analysis

This research used 25 open-source Java projects taken from http://sourceforge.net/, https://projectsgeek.com/, and http://www.enggroom.com/Project.aspx. Projects cover various applications, including core software development, internet gaming or entertainment, science or engineering, communications, network, and database.

Java projects containing the project version, the total Number of Java Files (NJF), and size in SLOC. We used the Automated POP Analyser (APA) Tool for software engineering for software metrics and size estimation for each Java system. The JAVA-language APA platform and framework function for JAVA-based projects. This method understands the java files to evaluate their source code for the details extracted. The POP value is determined based on every project file. The project sources used for analysis are summarised in Table 3.

Table 3. Java projects used in a case study.

No.	Project Name	Project Sources	NJF	SLOC
1	Civilization Game Project	https://projectsgeek.com/2016/01/civilisation-game-project-in-java.html	17	2559
2.	MESP 1.0	https://sourceforge.net/projects/expression-tree/	50	1189
3.	Jmol	https://sourceforge.net/projects/jmol/	156	22686
4.	JDMP	https://jdmp.org/	92	2684
5.	Geometry library (gpcj-2.1.0)	https://sourceforge.net/projects/gpcj/	12	2893
6.	Geometry library (gpcj-2.1.2)	https://sourceforge.net/projects/gpcj/	14	1995
7.	Intranet	https://projectsgeek.com/2014/07/intranet-mailing-system-project-java.html	9	1977
8.	**JavaCallTracer**	**https://sourceforge.net/projects/javacalltracer/**	**5**	**102**
9.	Java AIMBot -1.4	https://sourceforge.net/projects/jaimbot/	30	4413
10.	javaGeom-0.11.0	https://sourceforge.net/projects/geom-java/	124	4098
11.	javaGeom-0.11.1	https://sourceforge.net/projects/geom-java/	123	4059
12.	Java MP4Box Gui -1.6	https://sourceforge.net/projects/javamp4boxgui/	10	255
13.	Java MP4Box Gui -1.7	https://sourceforge.net/projects/javamp4boxgui/	15	404
14.	Java MP4Box Gui -1.8	https://sourceforge.net/projects/javamp4boxgui/	16	486
15.	JDistlib-0.0.7	http://jdistlib.sourceforge.net/	44	4578
16.	Simplified encryption (jasypt-1.9.1)	https://sourceforge.net/projects/jasypt/	115	7264
17.	iText® (iText5.4.0)	https://sourceforge.net/projects/itext/	96	3651
18.	iText® (iText5.4.3)	https://sourceforge.net/projects/itext/	100	3582
19.	JGraphT-0.8.2	https://sourceforge.net/projects/jgrapht	178	644
20.	**JGraphT-0.8.3**	**https://sourceforge.net/projects/jgrapht**	**180**	**644**
21.	HBX Binaural Player(HBX-1.16)	https://sourceforge.net/projects/hbxplayer/	11	2837
22.	HBX Binaural Player(HBX-1.16.1)	https://sourceforge.net/projects/hbxplayer/	11	2837
23.	iText® (iText5.3.0)	https://sourceforge.net/projects/jgrapht	37	1276
24.	Jaimlib-0.5	https://sourceforge.net/projects/jaimlib	45	1539
25.	HBX Binaural Player (HBX-1.15)	https://sourceforge.net/projects/hbxplayer/	9	2308

5.1. Results of POP, Refined POP, and Effort Assessment

We used the Automated POP Analyser (APA) Tool and Metric Investigation Tool CCCC by Littlefair [23]. for the assessment of various metrics and POP values for every individual java file of each Java project by following the procedure discussed in section 3.1 and 3.2 and then taking an average of the values for determining AvgDIT, AvgNOC, and AvgWMC values. The software effort estimation is done according to the COCOMO-I model of organic type. Various values, including that of POP, refined POP, and Effort calculated, are given in Table 4.

Table 4. The Value of POP metric estimation.

No.	TLC	AvgNOC	AvgDIT	AvgWMC	Total POP	Total Refined POP	Effort Calculated
1	18.0	12.0	5.166	70.88	510.99	318.34	6.02
2.	44.0	44.0	22.0	44.21	707.64	472.85	2.43
3.	165	100.99	46.99	67.08	3611.53	2374.25	50.87
4.	107.0	74.0	33.03	55.04	1870.85	1183.60	5.67
5.	12	8.3	4.0	96.74	449.87	336.18	6.76
6.	10.0	7.5	3.5	66.91	129.97	87.55	4.66
7.	21.0	10.28	3.81	36.09	350.46	231.17	4.56
8.	2.0	2.0	1.0	41.91	20.10	13.43	0.20
9.	33.0	26.5	12.03	81.54	1000.87	674.17	9.78
10.	145.0	84.0	34.25	76.76	4306.95	2645.90	8.61
11.	153.0	87.0	36.37	76.02	4676.44	2850.37	8.70
12.	4.0	3.0	1.33	41.91	44.22	29.55	0.51
13.	8.0	7.0	3.33	48.19	48.24	33.24	0.83
14.	8.0	7.0	3.33	40.16	44.22	29.55	1.01
15.	4.0	2.0	0.75	63.11	265.31	160.86	14.69
16.	86.0	75.0	35.66	68.60	2627.54	1713.99	15.71
17.	75.0	62.0	29.31	60.63	1195.90	791.48	7.72
18.	78.0	64.0	30.28	58.21	1191.87	788.80	7.55
19.	15.0	11.0	4.83	32.08	226.01	143.10	1.33
20.	15.0	11.0	4.83	32.08	226.01	143.10	1.33
21.	12.0	8.0	3.73	94.14	528.20	348.83	6.51
22.	12.0	8.0	3.73	94.14	528.20	348.83	6.51
23.	24.0	23.0	11.33	73.80	385.98	257.92	2.69
24.	44.0	35.0	16.0	48.89	910.79	586.05	3.17
25.	11.0	7.0	3.23	89.64	462.13	304.13	5.30

In the current case study, 25 Java projects used for the validation process have been maintained in two different lists based on even and odd projects, as depicted in Equations (5) and (6), respectively, and effort assessment through POP and refined POP are calculated by taking ratios of the respective projects in the two lists. A comparison of efforts assessment through POP and Refined POP with the actual Effort is shown in Table 5.

$$A = (E_1\ E_2 \dots \dots \dots \dots \dots \dots \dots \dots \dots E_n) \quad (5)$$

$$B = (O_1\ O_2 \dots \dots \dots \dots \dots \dots \dots \dots \dots O_n) \quad (6)$$

Table 5. The Effort estimation measure through POP, Refined POP comparison with actual.

No.	Total POP Ratio	Total Refined POP Ratio	Effort Calculated Ratio
1	1.3848	1.4853	0.4036
2.	0.5180	0.4985	0.1114
3.	0.2889	0.2904	0.6893
4.	0.0573	0.0560	0.0438
5.	4.3032	3.9246	0.8803
6.	0.0094	0.0103	0.0586
7.	0.9166	0.9889	1.2168
8.	9.9036	9.6551	1.0694
9.	0.9968	0.9976	0.9779
10.	1	1	1
11.	1	1	1
12.	2.3596	2.272	1.1784

5.2. Comparison of Results of POP Metrics

The performance evaluation has been designed based on four different types of POP metric values. POP metric calculation based on the PRICE system, Minkiewicz [24], the overall project, each Java file of the project, and refined POP values. For the POP metric comparison, two different projects with the smallest and highest number of Java files depicted in bold in Table 3 have been considered. As per Figure 5, the refined POP is better than the other POP metrics.

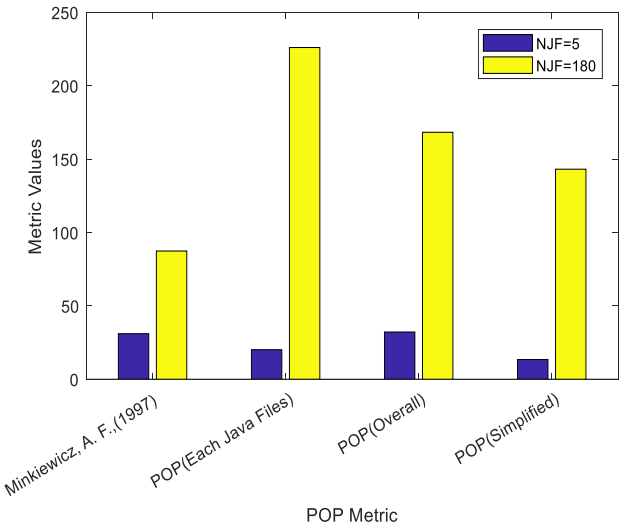

Figure 5. POP Count values based on two different projects.

The actual Effort calculated through COCOMO-I has a value of 0.1514 for the projects with NJF=5/NJF=180. The estimated effort values were assessed using different POP metrics like POP defined by Minkiewicz [24], POP (Overall), POP (Each Java Files), and POP (Refined) are 0.3548, 0.1911, 0.088, and 0.0938 respectively. The POP (Refined) value is much closer to the actual effort, as shown in Figure 6.

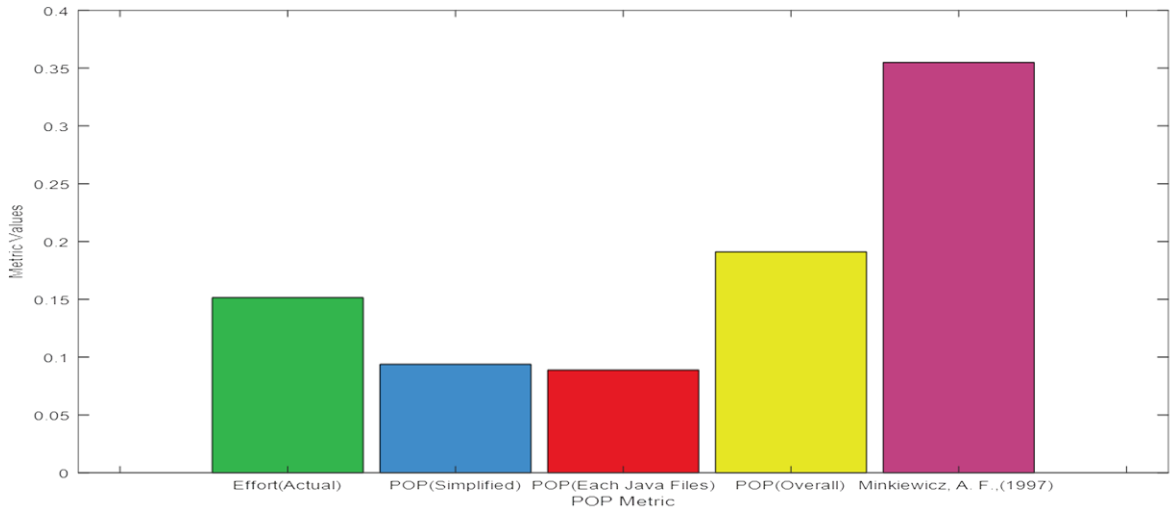

Figure 6. Efforts comparison based on two java projects.

5.3. Analysis and Comparison of Results of Deep Learning

In the Regression analysis, the R-value calculates the interplay of outputs and goals. An R-value of 1 implies a close connection, and 0 is a random relation. The standard output feature for training feedback neural networks is the average number of network error squares. Regression plots for CFNN and MLP-based Effort comparison using LM and BR training algorithms have been shown in Figures 7 and 8, respectively.

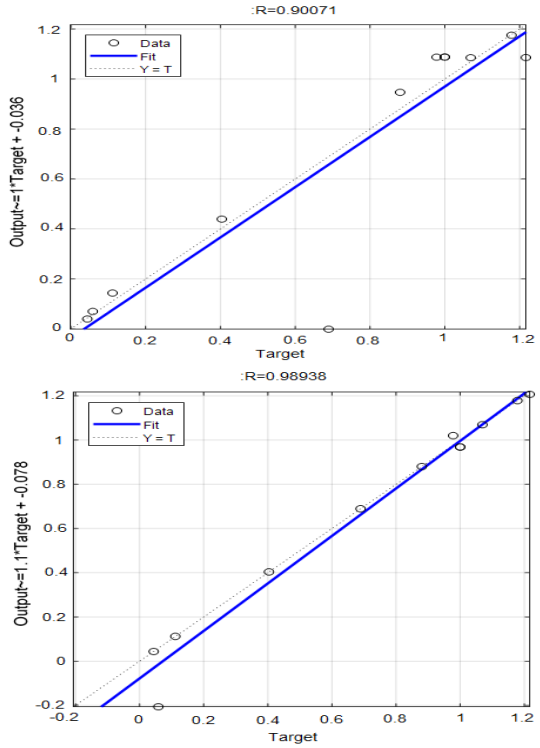

Figure 7. Regression plot for CFNN generated for Efforts comparison based on LM and BR, respectively.

The correlation between the actual and predicted value in both the training and testing phase is depicted with the correlation coefficient (R2) as 0.90071 (CFNN with LM), 0.9893 (CFNN with BR), 0.9370 (MLP with LM), and 0.9985 (MLP with BR) respectively in the Figures 7 and 8. These figures clearly show that the Effort estimated based on the refined POP metric is very close to the actual Effort. The R-value for MLP with BR has the highest value as 0.9985, indicating that it is almost 99.85 % closer to real Effort and has only a 0.0015% error in prediction.

The MSE represents the average square difference between the actual output and the goal. The lowest value of MSE indicates no error. Figures 9 and 10 display the average MSE in the prediction of the OO Metric, which is very low. The deep learning for enhanced OO software metrics has given outstanding performance with 99.85% accuracy. Figure 11 gives the performance comparison of MLP & CFNN based on LM and BR

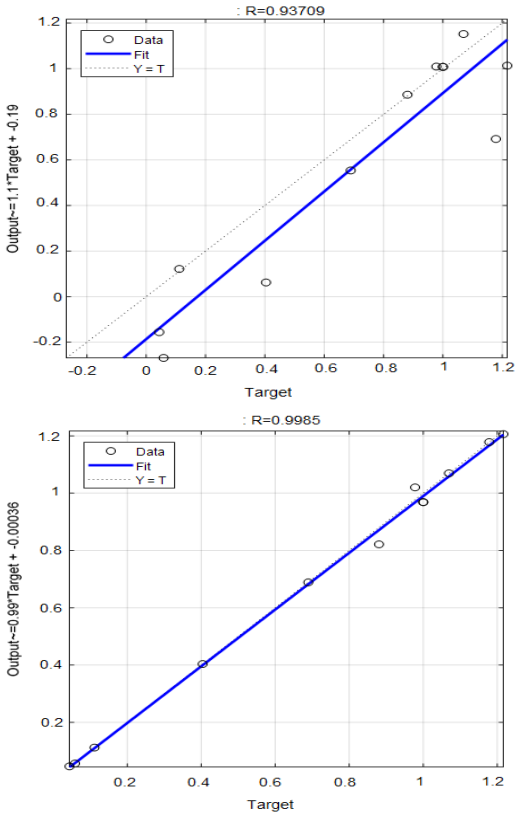

Figure 8. Regression plot for MLP generated for Efforts comparison based on LM and BR, respectively.

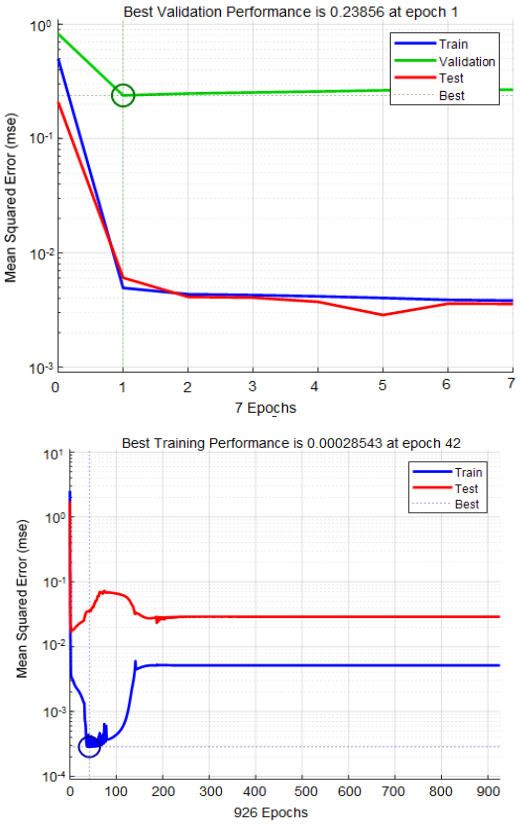

Figure 9. MSE for CFNN-based Effort comparison using LM and BR.

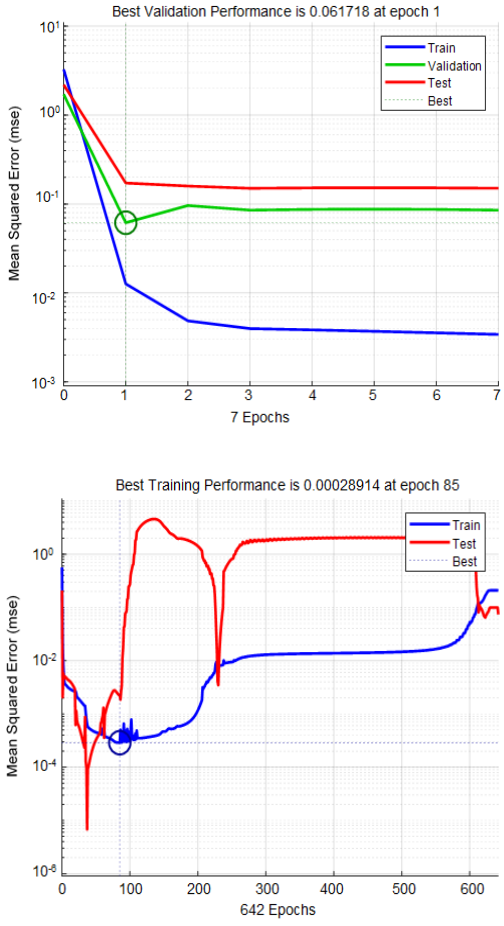

Figure 10. MSE for MLP-based Effort comparison based on LM and BR, respectively.

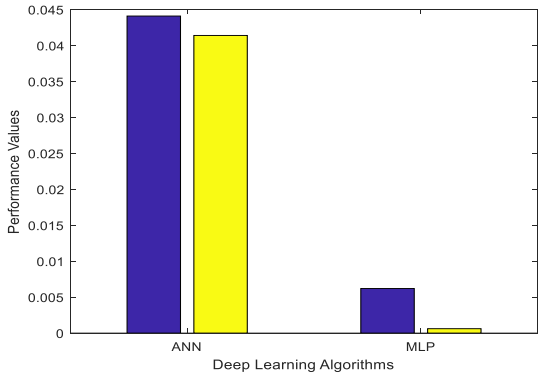

Figure 11. Performance comparison of MLP & CFNN based on LM (in blue) and BR (in Yellow).

6. Conclusions and Future Directions

For the 25 open-source Java projects, the original and refined POP metrics are calculated using the APA tool. The final result analysis was done based on various POP-based size metrics and calculating the efforts with these metrics. The POP metric is a good size measure of software which can be easily seen from the results.

We used the POP metrics to estimate the development efforts through deep learning methods with different training models. The Effort estimated through the refined POP metric and various other forms of the POP metrics using deep learning methods was analysed and compared with the actual Effort calculated through the COCOMO model.

The highest correlation coefficient value between the actual Effort and the Effort predicted through refined POP using MLP with BR, and the minimum MSE value establishes the deep learning-based approach of effort estimation through refined POP using MLP with BR algorithm is the best.

This research used Java programming language-based projects to calculate POP and refined POP metrics for effort estimation.

The Refined POP metric was validated by comparing the effort values estimated through this metric and the SLOC metric using the COCOMO-I model. The work can be extended for the object-oriented-based software written in some other programming language. Also, in place of the static regression model used by us, the dynamic Machine learning-based models can be used.

References

[1] Abdallah M. and Alrifaee M., "A Heuristic Tool for Measuring Software Quality Using Program Language Standards," *The International Arab Journal of Information Technology*, vol. 19, no. 3, pp. 314-322, 2022.

[2] Aggarwal K., Singh Y., Kaur A., and Malhotra R., "Software Reuse Metrics for Object-Oriented Systems," *in Proceedings of the 3rd ACIS Int'l Conference on Software Engineering Research,*

Management and Applications, Mount Pleasant, pp. 48-54, 2005.

[3] Akinsanya B., Araújo L., Charikova M., Gimaeva S., Grichshenko A., Khan A., Mazzara M., Okonicha N., and Shilintsev D., "Machine Learning and Value Generation in Software Development: a Survey," *in Proceedings of the International Conference on Tools and Methods for Program Analysis*, Tbilisi, pp. 44-55, 2020.

[4] Albrecht A., Measuring Application Development Productivity, *in Proceedings of the Joint Share, Guide, and IBM Application Development Symposium*, Monterey pp. 14-17, 1979.

[5] Albrecht A. and Gaffney J., "Software Function, Source Lines of Code, and Development Effort Prediction: a Software Science Validation," *IEEE Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 639-648, 1983.

[6] Chidamber S. and Kemerer C., "Towards a Metrics Suite for Object-Oriented Design," *in Proceedings on Object-Oriented Programming Systems, Languages, and Applications*, United States, pp. 197-211, 1991.

[7] Chidamber S. and Kemerer C., "A Metrics Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476-493, 1994.

[8] Chidamber S., Darcy D., and Kemerer, C., "Managerial Use of Metrics for Object-Oriented Software: an Exploratory Analysis," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 629-639, 1998.

[9] De Barcelos Tronto I., Da Silva J., and Sant'Anna N., "Comparison of Artificial Neural Network and Regression Models in Software Effort Estimation," *in Proceedings of the International Joint Conference on Neural Networks*, Orlando, pp. 771-776, 2007.

[10] Fenton N. and Bieman J., *Software Metrics: A Rigorous and Practical Approach*, CRC Press, 2014.

[11] Finschi L., "An Implementation of the Levenberg-Marquardt Algorithm," Eidgenössische Technische Hochschule Zürich, 1996.

[12] Ghatasheh N., Faris H., Aljarah I., and Al-Sayyed M., "Optimising Software Effort Estimation Models Using Firefly Algorithm" *arXiv preprint arXiv:1903.02079*, 2019.

[13] Haykin S., *Neural Networks, A Comprehensive Foundation*, Prentice-Hall, 1999.

[14] Heiat A., "Comparison of Artificial Neural Network and Regression Models for Estimating Software Development Effort," *Information and Software Technology*, no. 44, vol. 15, pp. 911-922, 2002.

[15] Haug M., Olsen E., and Bergman L., *Software Process Improvement: Metrics, Measurement, and Process Modelling: Software Best Practice 4*, Springer Science and Business Media, 2011.

[16] Idri A., Khoshgoftaar T., and Abran A., "Can Neural Networks Be Easily Interpreted in Software Cost Estimation?" *in Proceedings of the IEEE World Congress on Computational Intelligence*, Honolulu, pp. 1162-1167, 2002.

[17] Jain S., Yadav V., and Singh R., "Assessment of Predictive Object Points (POP) Values for Java Projects," *International Journal of Advanced Computer Research*, vol. 3, no. 4, pp. 298-300, 2013.

[18] Jørgensen M., "A Review of Studies on Expert Estimation of Software Development Effort," *Journal of Systems and Software*, vol. 70, no. 1-2, pp. 37-60, 2004.

[19] Kanmani S., Kathiravan J., Kumar S., and Shanmugam M., "Neural Network-Based Effort Estimation Using Class Points for OO Systems," *in Proceedings of the International Conference on Computing: Theory and Applications*, Kolkata, pp. 261-266, 2007.

[20] Judge T. and Williams A., "Oo Estimation-An Investigation of the Predictive Object Points (POP) Sizing Metric in an Industrial Setting," Parallax Solutions Ltd, Coventry, UK, 2001.

[21] Khalid A., Latif M., and Adnan M., "An Approach to Estimate the Duration of Software Project through Machine Learning Techniques," *Gomal University Journal of Research*, vol. 33, no. 1, pp. 47-59, 2017.

[22] Khoshgoftaar T., Allen E., Hudepohl J., and Aud S., "Application of Neural Networks to Software Quality Modelling of a Very Large Telecommunications System," *IEEE Transactions on Neural Networks*, vol. 8, no. 4, pp. 902-909, 1997.

[23] Littlefair T., CCCC Metric Tool, Available from: http://www.fste.ac.cowan.edu.au/~tlittlef/, Last Visited, 2022.

[24] Minkiewicz A., "Measuring Object-Oriented Software with Predictive Object Points," PRICE Systems, LLC, pp. 609-866, 1997.

[25] Minkiewicz A. and Fad B, and Lockheed Martin Corp, "Parametric Software Forecasting System and Method," U.S. Patent 6,073,107, 2000.

[26] Nassif A., Azzeh M., Idri A., and Abran A., "Software Development Effort Estimation Using Regression Fuzzy Models," *Computational Intelligence and Neuroscience*, vol. 2019, 2019.

[27] Regolin E., de Souza G., Pozo A., and Vergilio S., "Exploring Machine Learning Techniques for Software Size Estimation," *in Proceedings of the 23rd International Conference of the Chilean Computer Science Society*, Chillan, pp. 130-136, 2003.

[28] Singh A., Bhatia R., and Singhrova A., "Taxonomy of Machine Learning Algorithms in

Software Fault Prediction Using Object-Oriented Metrics," *Procedia Computer Science*, vol. 132, pp. 993-1001, 2018.

[29] Wittig G. and Finnic G., "Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort," *Australasian Journal of Information Systems*, vol. 1, no. 2, 1994.

[30] Verner J. and Tate G., "A Software Size Model," *IEEE Transactions on Software Engineering*, vol. 18, no. 4, pp. 265-278, 1992.

[31] Yadav V., Yadav V., and Singh R., "Introducing New OO Metric for Simplification in Predictive Object Points (POP) Estimation Process in OO Environment," *International Journal of Engineering Sciences and Research Technology*, vol. 5, no. 1, pp. 716-723, 2016.

[32] Zhang D. and Tsai J., "Machine Learning and Software Engineering," *Software Quality Journal*, vol. 11, no. 2, pp. 87-119, 2003.

Vijay Yadav he is B.Tech Hons in (IT) from CSJM University Kanpur (UIET) and M.Tech Hons in (CSE) from UP Technical University. Currently, he is doing a Ph.D. in (CSE) from Dr. A.P.J. Abdul Kalam Technical University, Uttar Pradesh. He has undergone projects like the Online Entertainment world and Enterprise Resource Planning System during his B.Tech (IT) curriculum. He has done his M.Tech (CSE) thesis in Object-Oriented Software Metrics (POP Automation). He is a meritorious B.Tech (IT) and M.Tech (CSE) student and has a merit excellence award. He has 9 papers in International Conferences/Journals, and two IEEE conferences attended for paper presentations.

Raghuraj Singh he is a B.Tech. (CSE), MS (Software Systems) and PhD in Computer Science and Engineering from UP Technical University. He has about 23 years of experience in teaching. Currently, he is HOD CSE in HBTU Kanpur. He has guided 7 PhDs and 17 M.Techs, and several B.E./B.Tech projects. He is the Chairman of Kanpur Chapter, CSI, Life Member of ISTE, Member of the Institution of Engineers (India), Fellow Member of IETE, Professional member of ACM, and Senior Member of the International Association of IACSIT. To his credit, he has more than 80 papers in National / International Conferences and Journals. Currently, 4 students are working for PhD, and 4 are pursuing M.Tech. under his guidance.

Vibhash Yadav He is a B.Tech. (CSE) from CSJM University, Kanpur, M.Tech in (CSE) from Maharshi Dayanand University, Rohtak, and PhD in Computer Science and Engineering from Uttrakhand Technical University, Dehradun. He has about 10 years of experience in teaching. He is an Associate Professor and Head in the IT Dept of Rajkiya Engineering College, Banda. He has guided 4 M.Techs and several B.E./B.Tech projects. He is the Member, Kanpur Chapter, CSI. He has more than 15 papers in National / International Conferences and Journals to his credit. Currently, 4 students are working for Ph.D. and 4 are pursuing M.Tech. under his guidance.