International Journal of Systems Signal Control and Engineering Application 10 (1-6): 24-31, 2017
ISSN: 1997-5422


# An Effectual Scheme to Improve COCOMO|| Model using OOPS Metric-Based Source Code Size

Sepide Sabrjoo
Department of Computer Engineering and Information Technology,
Payam-e-Noor University, Tehran, Iran

**Abstract:** Now a days, the correct estimation of effort, cost and time for the process of software development plays a major role in the success or failure of software engineering projects such that the acquisition of project size is the first step to estimate the effort of software. COCOMO|| Model is the clearest and most reliable model for the estimation of the costs of software. Results from previous studies indicate that because of different structures in models, changes in the proposed hypotheses with the passage of time and different estimations in project size, the difference between predicted and real values are huge. In this study, an effective scheme is proposed which improves the time, cost and effort in the software for the COCOMO|| Model. The proposed scheme uses Object-Oriented Project Size (OOPS) metric to decrease the errors of source-code size estimations. The OOPS used here is extracted from the class diagram of the project. The obtained experimental results from 12 Java projects show that the proposed method satisfies the COCOMO|| Model properties as well and decreases the Mean Magnitude of Relative Error (MMRE) compared to the other works.

**Key words:** Estimation, effort, reliable, indicate, experimental, obtained

## INTRODUCTION

Designing software systems are tough and expensive. Software engineering as a science, proposes the ways to measure a project in quantity (Saez *et al*., 2016). The reports from the projects indicate that there is almost no control on software projects, so that the real efforts of a software project are more than estimated efforts (Mittas *et al*., 2015). Therefore, projects generally last longer than the planned scheduled time (Ceke and Milasinovic, 2015). Thus, the estimation of time, cost and effort to fulfill the project and the factors is undoubtedly a very serious issue (Chu, 2016). In recent years, numerous studies have been conducted in this area, resulting in the increase of software estimation accuracy (Jain *et al*., 2014a, b). The estimation of software development cost is a major factor in project development which includes a variety of methods and techniques (Jorgensen and Shepperd, 2007; Xu and Khoshgoftaar, 2004). These techniques are included top-down, bottoms-up, expert judgment, Parkinson's Law, estimation based on analogy, function points and object points (Jorgensen and Sjoberg, 2004; Mittas *et al*., 2008). Among these models, the estimation of cost based on object points is more efficient because it does not rely on the details of implementation in which the estimation of complexity factor is simpler (Dio *et al*., 2015). Obtaining the size of a project is the first step in estimating the effort of software. Early estimation of code size has evolved as an important research issue in software science, because it enables software managers to ask the effort for the required development in the software projects in the early development phase (Azam *et al*., 2014; Pfleeger *et al*., 2005). Further, it assists the allocation of resources, efficient development and effective planning in development activities (Jorgensen and Shepperd, 2007).

SLOC (Source Lines of Code) as the input of early size, has been used in most of the cost estimation instruments such as COCOMO, COCOMO||, Price/S, SEER, SLIM, etc. (Zhou *et al*., 2014). Each of these models has their own advantages and disadvantages. For example, SEER-SEM has two main limitations in effort estimation. First, there are over fifty input parameters related to the various factors of software projects which might increase the complexity of SEE-SEM, especially for managing the uncertainty from these inputs. Second, the specific details of SEER-SEM increase the difficulty of discovering the nonlinear relationship between the parameter inputs and the corresponding outputs (Dio *et al*., 2015).

The second of Constructive Cost Model (COCOMO||) is the clearest existing model to estimate software development cost (Jain and Singh, 2014a, b; Chalotra *et al*., 2015). The model is better than the other models because of several reasons: it provides sufficient documents for people that various commercial instruments are available to use it, it is widely evaluated and used in different organizations, it is an empirical model which has been acquired through collecting data from different software projects. Documents of this model are available and used in most of the organizations. The model is implemented via different algorithms (Soleimanian *et al*., 2015). Antoniol *et al*. (1999) COCOMO 81 Model has been proposed by Boehm. COCOMO uses parameters for software effort estimation which are calculated by regression analysis of 63 types of project data. This version assumes that the software under design is produced based on waterfall model and is implemented using the structured languages such as C or FORTRAN. COCOMO Model exists in a basic, intermediate and advanced form. The model considers cost, features of project, product, hardware and staffs in a precise estimation. However, while it works properly for existing software projects, it faces problems with regard to new software methods (Boehm, 2000). The COCOMO|| supports a spiral model in which source code size is assumed as a key input. The results from independent studies on COCOMO|| Models indicate that there is a significant difference between the predicted values and the real value. The reason could be one of the following: Different structure of models, change in the proposed hypotheses by the passage of time, wrong estimation of project size (Johnson, 1998; Chen *et al*., 2004). The proposed method by Garg *et al*. (2014) provides better results from COCOMO|| Model. In this method, COCOMO|| Model has been used at the middle level. Fifteen extra cost drivers are the new features of this model. These cost drivers transform the values which are required to get multiplication factor modulator of the estimation to a constant value (EAF). In this method, COCOMO|| Model is integrated with the Function Point (FP) Model so as to reduce the complexity of the model, whereby accuracy has is improved in COCOMO|| Model by means of ratio of function points to Kilo Lines of Code (KLOC). However, the calculation of FP has an extent of change. The rules for FP calculation have been defined and formulated properly but FP manual count and recount processes are more expensive and more time-consuming than automatic count. Also, in the method MMRE parameter is 0.2641.

Soleimanian *et al*. (2015) a hybrid of Genetic algorithm with a Tabu search algorithm is used for effort estimation. Although, in this model, the effort estimated in COCOMO|| Model is improved but the execution time is long and in high computing becomes problematic.

According to Soleimanian *et al*. (2015), COCOMO|| Model is improved by a continuous genetic algorithm. In this method, 60 projects from the dataset of NASA projects have been selected to study the efficiency of the model. The results show that this algorithm is capable of moderating the required parameters of the COCOMO|| Model and creating an effective model in SCE of COCOMO||. However, MMRE parameter in this method is 0.2153 which is greater than 0.2 and is not very accurate.

In this study, we have tried to partially improve the size parameter in the COCOMO|| Model. In development of object-oriented software, class diagrams are available in the step of early development, found as the basis for producing source code in the system. Thus, it is reasonable to use information obtained from class diagrams to estimate SLOC in object-oriented system (e.g., POPS and OOPS metric). The POPS (Predictive Object Points) metric is a suitable metric to estimate software size and evaluate the effort and find the cost and the project timetable which is based on a behavior which proposes each class together with high-class inputs to define the structure of a system (Leung and Fan, 2002). This metric is a suitable size index in object-based systems. Investigations indicate that POPS metric outperforms FP metric in size estimation which can have the best application in the estimation of object-oriented systems (Jain *et al*., 2014a, b). This metric has been used by Jain and Singh (2014a, b) to estimate the required effort in COCOMO|| Model which is used in KLOC measurement and effort estimation using linear regression model (Zhou *et al*., 2014) the researchers proposed a source code prediction model based on OOPS (Object-Oriented Project Size) metric. In UML-Based Software Sizing, yet to date, POPS metric has been used in COCOMO|| Model. The analysis of scientific investigations shows that very few works have conducted a comprehensive and comparative analysis on estimated SLOC based on OOPS and POPS metric. Moreover, OOPS metric is yet to be used in COCOMO|| Model.

In this study, an effective scheme has been proposed to improve COCOMO|| Model in the estimation of cost, time and effort of software which uses source code size based on OOPS metric extracted from class diagram. The obtained experimental results from 12 Java projects and their API files show that the proposed method satisfies the COCOMO|| Model properties as well and decreases the Mean Magnitude of Relative Error (MMRE) compared to the other works. This information is extracted from SCI tools that are available in the development step of software.

## Basic theory

**COCOMO|| Model:** In this model, effort of software is acquired from Eq. 1 to complete the project based on the person and month through calculating the project size (Garg *et al*., 2014):

$$\text{Effort} = A\times(\text{size})\ B\times PM \quad (1)$$

In Eq. 1, PM is the effort adjustment factor for effort estimation which is considered equal to 1 for ease of comparison in all the projects; A and B refer to the coefficients which are calculated in three different modes of COCOMO|| Model regarding (Table 1) (Jain and Singh, 2014a, b).

**OOPS metric:** This metric is a size metric in object-oriented software in which the names of class, attributes, methods and parameters are defined. The metric is calculated as follows (Zhou *et al*., 2014):

- OOPS = 0, Token set = {}
- Name of process class; if the name of the class does not exist in Token set per token, it will be added to it, whereby OOPS will equal to 1
- Attributes of process in class; if the character does not exist in Token set per token, the token will be added to Token set and the attributes will be added to OOPS
- Process methods in class; if the name of the method does not exist in Token set per token, the token will be added to Token set and the number of parameters will be added to OOPS

Table 1: The values of A and B in COCOMO|| Model (Johnson, 1998)

| Models | A | B | Project size |
|---|---|---|---|
| Organizational | 2.4 | 1.05 | The value of KLOC ranges from 2-50 |
| Semi-open | 3.0 | 1.12 | The value of KLOC ranges from 50-300 |
| Embedded | 3.6 | 1.20 | The value of KLOC exceeds from 300 |

Although, the determination of a proper token is tough in most cases in this metric but OOPS is acquired from class diagram. Therefore, it is used to predict SLOC at the earliest step of development.

**Pops metric:** This metric is a suitable metric to estimate software size which is grounded in behavioral basis, proposed each class together with high-level inputs to define the structure of a system and calculated based on the equation (Jain and Singh, 2014a, b):

$$\text{Pops} = \frac{(1+\left[(1+\text{Avgnoc})\times\text{AvgDIT}\right]^{1.01} + \left(\left|\text{AvgNOC}-\text{AvgDIT}\right|\right)^{0.01})\times\text{TIC}\times\text{WMC}\times\text{AMC}}{7.8} \quad (2)$$

According to the equation above, AVGNOC equals to average number of classes that inherit from a class directly; AVGDIT equals to average depth of inheritance tree (it is calculated based on class inherit and index); AVGTLC equals to average number of zero-level classes in class diagram (it is calculated based on class inherit and index); AMC equals to average number of methods in each class; WMC equals to average number of weighted methods in each class.

## MATERIALS AND METHODS

**The proposed method:** The proposed method has been shown in Fig. 1 using block diagram. The effort estimation

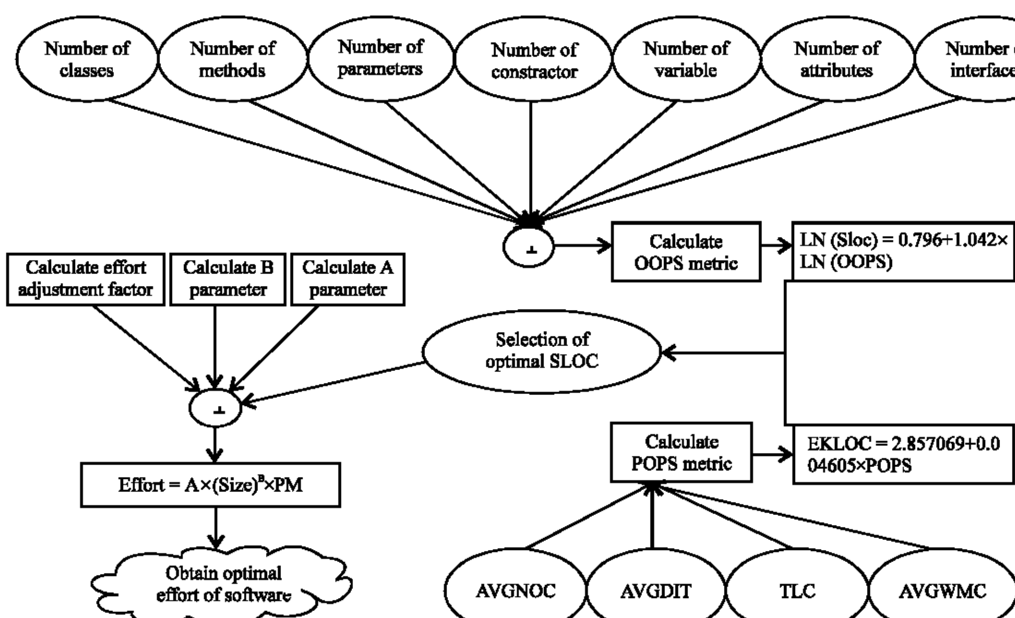


Fig. 1: Block diagram of the proposed method

Table 2: Measurement of SLOC based on OOPS and POPS metric

| Project name | OOPS count | POPS count | Actual SLOC | ESLOC based on OOPS | ESLOC based on POPS |
|---|---|---|---|---|---|
| abbot-1.3.0 | 1367 | 51.5141 | 4326 | 4104 | 3094 |
| barcode4j-2.1.0 | 740 | 1.4370 | 1863 | 2164 | 2863 |
| ftp4j-1.7.2 | 2460 | 1008 | 10488 | 7569 | 7493 |
| httpunit-1.7 | 18227 | 17653 | 70150 | 61004 | 84153 |
| jgap_3.6.2_full | 27477 | 19140 | 147639 | 93564 | 91000 |
| jip-src-1.2 | 8979 | 3500 | 34993 | 29172 | 19000 |
| krysalis-jCharts-1.0.0-alpha-1 | 5655 | 3120 | 21996 | 18020 | 17224 |
| mx4j-3.0.2 | 1547 | 18.1 | 5055 | 4700 | 3000 |
| openfast-1.1.2 | 6367 | 2788 | 22726 | 20389 | 16000 |
| PDFBox-0.7.3 | 14962 | 14164 | 65494 | 49664 | 68082 |
| prevayler-2.3 | 2800 | 2000 | 9256 | 8663 | 12060 |
| xBaseJ | 4998 | 3900 | 26473 | 15844 | 20810 |

scheme to improve COCOMO|| Model consists of 7 steps. The block diagram of the proposed approach is shown in Fig. 1 and it is detailed as follows:

- At the first step, the number of classes, methods, parameters, variables, constructor, attributes and interfaces of each class is measured regarding OOPS calculation method
- In the second step, the value of SLOC is calculated based on metric OOPS
- At this step, parameters AVGNOC, AVGDIT, TLC and AVGWMC are measured to calculate POPS metric
- At this step, the value of SLOC is calculated based on POPS metric
- At this step, the value of actual SLOC is compared with calculated SLOC and then it is specified based on which metric, SLOC estimation has more optimum values, after calculating the mean magnitude of relative error
- The value of A, B and PM are calculated for each project regarding project size
- The required effort in software is estimated by substituting optimal values of SLOC as A, B and PM in COCOMO|| Model

## RESULTS AND DISCUSSION

**Optimal estimation of source code size:** In this study, 12 JAVA projects which are available as open source at www.sourceforge.net together with their API files which are generally the documents related to classes, attributes and so forth, on the step of software development have been examined. In addition, SciTools (ver.3.1) has been used to measure and analyze the required parameters. As shown in Table 2, the required parameters to estimate OOPS which includes the number of attributes, methods, parameters and variables of class have been estimated using SciTools and the number of classes and the relationship between them have been extracted from class hierarchy file, whereby the value of SLOC has been calculated by putting the value of calculated OOPS in the optimal equation for SLOC prediction which is represented as follow (Zhou *et al.*, 2014):

$$\text{LN (ESLOC)} = 0.796+1.042\times \text{LN (OOPS)} \tag{3}$$

At the next step, to estimate POPS metric, AVGNOC, AVGDIT, TLC and AVGWMC parameters have been calculated using the hierarchy class and ultimately the value of SLOC has been calculated for each of them based on the equation (Jain *et al.*, 2014):

$$\text{EKLOC} = 2.857069+0.004605\times \text{POPS} \tag{4}$$

According to Eq. 4, EKLOC represents the estimated kilo source lines of source code. The results are shown in Table 2. The name of projects under study is represented in Column 1 of the table, an estimation of OOPS metric and ESLOC (Estimated Source Lines of Code) value estimated based on OOPS metric has been represented in Column 2 and 5 and estimation of the value of POPS metric and ESLOC estimated based on POPS metric is represented in Column 3 and 6 and size of actual SLOC has been represented in Column 4.

**Optimal estimation of effort in COCOMO|| Model:** According to Table 2, in the majority of evaluating projects, ESLOC measuring based on OOPS metric is closer to the real size. Therefore, we use their values in the estimation of software effort in COCOMO|| Model and after assigning the values of A and B based on the source code to their size; the software effort required is calculated according to Eq. 1. The results can be seen in Table 3.

**Comparison of COCOMO|| Model based on source code size with previous works:** In this study of the research, we compare the proposed method with previous works. The proposed model is a more optimal model than early COCOMO|| Model because the early COCOMO|| Model is suitable for older software projects. Yet, it has faced problems in new methods for producing software. Further, In COCOMO|| Model (Boehm, 2000), 17 cost drivers are used to estimate cost of software. This number and the factors are increased in cost estimation in the improved model. The improved COCOMO|| Model (Garg *et al.*, 2014) is used at the middle level those 15 extra cost drivers are the new feature of this model than previous COCOMO|| Model. In this method, COCOMO|| Model is integrated with the Function Point (FP) Model so as to reduce the complexity of the model, whereby the accuracy has increased in COCOMO|| Model by means of ratio of function points to Kilo Lines of Code (KLOC). But FP manual count and recount processes are more expensive and more time-consuming than the automatic count. Yet, knowing software size before producing it can be helpful in these estimations. There are a variety of methods to acquire software size. Yet, all of the methods have numerous weaknesses such as mismatch with variety types of software, hardness of calculation and dependency on technology. Therefore, a majority of software professors and experts have made an attempt to find a simple method and standards to measure modern software. Jain and Singh (2014a, b), POPS metric has been used to estimate source code lines in COCOMO|| Model which has given optimal values to date. To compare this method with the proposed method, two function measures are used to measure the accuracy of COCOMO|| Model. MAR (Mean of Absolute Residuals) and MMRE (Mean Magnitude of Relative Error) are based on real and predicted values. In this study, any data point is corresponding to a system under study. Yi is the source code size and the amount of real effort per given point i. y^i is the source code size and the amount of predicted effort based on proposed method and (Jain and Singh, 2014a, b), so that $1 \leq i \leq 12$. Thus, AR, MRE, MAR and MMRW are as follows per data point (i). The results can be seen in Table 4 and 5.

Table 3: Optimal estimation of effort calculated in the proposed method

| Project name | A | B | ESLOC | Effort |
|---|---|---|---|---|
| abbot-1.3.0 | 2.4 | 1.05 | 4104 | 10.57 |
| barcode4j-2.1.0 | 2.4 | 1.05 | 2164 | 5.3979 |
| ftp4j-1.7.2 | 2.4 | 1.05 | 7569 | 20.1 |
| httpunit-1.7 | 3 | 1.12 | 61004 | 299.72 |
| jgap_3.6.2_full | 3 | 1.12 | 93564 | 483.9 |
| jip-src-1.2 | 2.4 | 1.05 | 29172 | 82.87 |
| krysalis-jCharts-1.0.0-alpha-1 | 2.4 | 1.05 | 18020 | 49.975 |
| mx4j-3.0.2 | 2.4 | 1.05 | 4700 | 12.18 |
| openfast-1.1.2 | 2.4 | 1.05 | 20389 | 56.89 |
| PDFBox-0.7.3 | 3 | 1.12 | 49664 | 238.060 |
| prevayler-2.3 | 2.4 | 1.05 | 8663 | 33.66 |
| xBaseJ | 2.4 | 1.05 | 15844 | 66.21 |

Table 4: MAR and MMRE of estimated COCOMO|| Model in proposed method

| | | | ESLOC in proposed method | | Effort calculated in proposed method | |
|---|---|---|---|---|---|---|
| Project name | Actual SLOC | Actual effort | $AR_i$ | $MRE_i$ | $AR_i$ | $MRE_i$ |
| abbot-1.3.0 | 4326 | 11.1658 | 222 | 0.0513 | 0.5958 | 0.0534 |
| barcode4j-2.1.0 | 1863 | 4.6124 | 301 | 0.1615 | 0.7855 | 0.1703 |
| ftp4j-1.7.2 | 10488 | 28.3099 | 2919 | 0.2783 | 8.2099 | 0.2900 |
| httpunit-1.7 | 70150 | 350.4872 | 9146 | 0.1303 | 50.7672 | 0.1448 |
| jgap_3.6.2_full | 147639 | 806.5410 | 54075 | 0.3662 | 322.6410 | 0.4000 |
| jip-src-1.2 | 34993 | 100.3209 | 5821 | 0.1663 | 17.4509 | 0.1740 |
| krysalis-jCharts-1.0.0-alpha-1 | 21996 | 61.6129 | 3976 | 0.1807 | 11.6379 | 0.1889 |
| mx4j-3.0.2 | 5055 | 13.1558 | 355 | 0.0702 | 0.9758 | 0.0742 |
| openfast-1.1.2 | 22726 | 63.7617 | 2337 | 0.1028 | 6.8717 | 0.1078 |
| PDFBox-0.7.3 | 65494 | 324.5390 | 15830 | 0.2417 | 86.4790 | 0.2665 |
| prevayler-2.3 | 9256 | 24.8288 | 593 | 0.0640 | 1.6000 | 0.3557 |
| xBaseJ | 26473 | 74.8435 | 10629 | 0.4015 | 8.6335 | 0.1154 |

MAR: 8850, 43.6566; MMRE: 0.1845, 0.1951

Table 5: MAR and MMRE of estimated COCOMO|| Model by Jain and Singh (2014a, b)

| | | | ESLOC in Jain and Singh (2014a, b) | | Effort calculated in Jain and Singh (2014a, b) | |
|---|---|---|---|---|---|---|
| Project name | Actual SLOC | Actual effort | $AR_i$ | $MRE_i$ | $AR_i$ | $MRE_i$ |
| abbot-1.3.0 | 4326 | 11.1658 | 1232 | 0.2847 | 3.3088 | 0.2963 |
| barcode4j-2.1.0 | 1863 | 4.6124 | 1000 | 0.5367 | 2.6276 | 0.5697 |
| ftp4j-1.7.2 | 10488 | 28.3099 | 2990 | 0.2850 | 8.4099 | 0.2971 |
| httpunit-1.7 | 70150 | 350.4872 | 14003 | 0.1996 | 79.2428 | 0.2261 |
| jgap_3.6.2_full | 147639 | 806.5410 | 56639 | 0.3836 | 337.4610 | 0.4184 |
| jip-src-1.2 | 34993 | 100.3209 | 15993 | 0.4570 | 47.4909 | 0.4734 |
| krysalis-jCharts-1.0.0-alpha-1 | 21996 | 61.6129 | 4772 | 0.2169 | 13.9629 | 0.2266 |
| mx4j-3.0.2 | 5055 | 13.1558 | 2055 | 0.4065 | 5.5493 | 0.4218 |
| openfast-1.1.2 | 22726 | 63.7617 | 6726 | 0.2959 | 19.6517 | 0.3082 |
| PDFBox-0.7.3 | 65494 | 324.5390 | 2588 | 0.0395 | 14.3910 | 0.0443 |
| prevayler-2.3 | 9256 | 24.8288 | 2804 | 0.3029 | 23.9412 | 0.9643 |
| xBaseJ | 26473 | 74.8435 | 5663 | 0.2139 | 15.0165 | 0.2006 |

MAR: 9705, 47.5878; MMRE: 0.3018, 0.3706

$$AR_i = y_i + y\hat{}_i \quad (5)$$

$$MRE_i = \frac{|y_i + y\hat{}_i|}{y_i} \quad (6)$$

$$MAR = \frac{1}{n}\sum_{i=1}^{n} AR_i \quad (7)$$

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} MRE_i \quad (8)$$

As shown in the tables above, MAR and MMRE in the estimation of source code size in the proposed method equals to 8850 and 0.1845 and 9705 and 0.3018 by Jain and Singh (2014a, b). It could be concluded that the estimation of source code size in the proposed method compares to Jain and Singh (2014a, b) is closer to the real value under the same conditions. Further, MAR and MMRE in the required effort of software are calculated equal to 43.6566 and 0.1951 in the proposed COCOMO|| Model and 47.5878 and 0.3706 by Jain and Singh (2014a, b). Also, Garg *et al.* (2014) MMRE for 20 projects is 0.2641. Soleimanian *et al.* (2015) which uses a hybrid of Genetic algorithm with a Tabu search algorithm to calculate effort estimates in COCOMO|| Model, MMRE is equal to 0.2973. Also, Soleimanian *et al.* (2015) which use a continuous genetic algorithm MMRE is equal to 0.2153 whereby it can deduce that the required effort of software is closer to its real value in the proposed method resulting in the more optimal estimation of cost and time. Further in Fig. 2, the estimated effort based on source code size in the proposed method, real effort and estimated effort by Jain and Singh (2014a, b) are compared, observing that the estimation diagram in the proposed method is adjusted with the calculated real effort diagram. Figure 3 shows the MMRE parameter measurement chart in COCOMO|| effort estimation models. The results show that MMRE parameter in the proposed method is much lower than the others.

**The relationship between OOPS metric and effort:** According to the results the projects under study that there is an exponential relationship between OOPS and

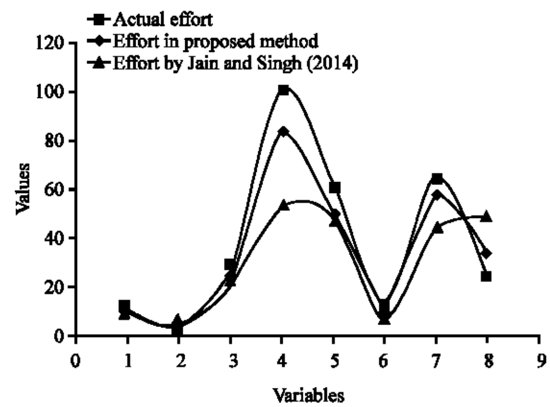


Fig. 2: Comparison of effort estimated based on source code size

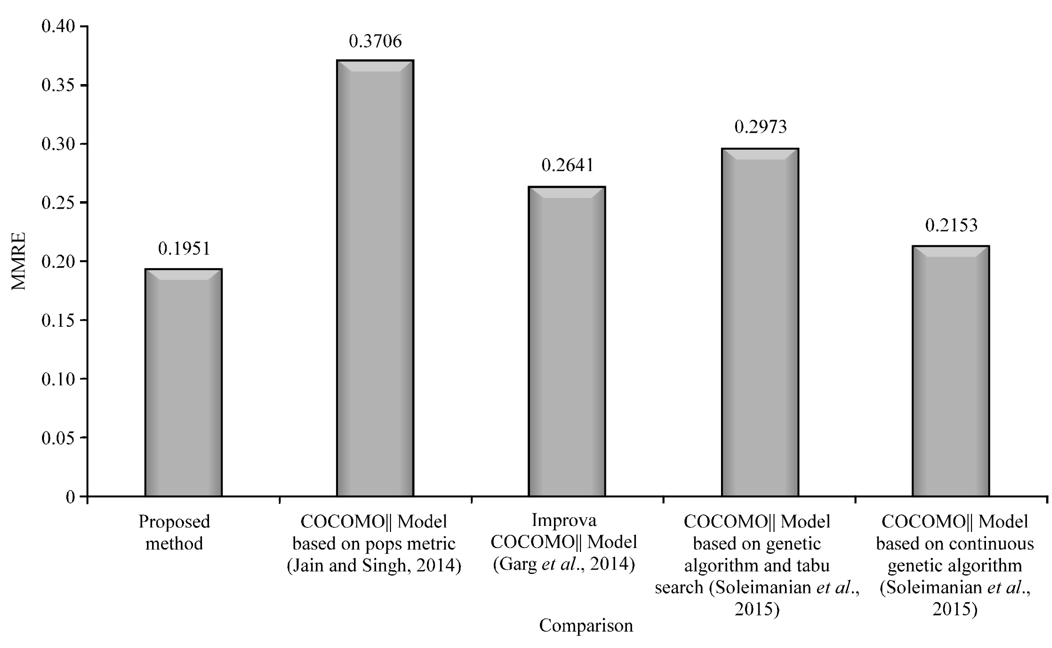


Fig. 3: Performance comparison of COCOMO|| based on MMRE

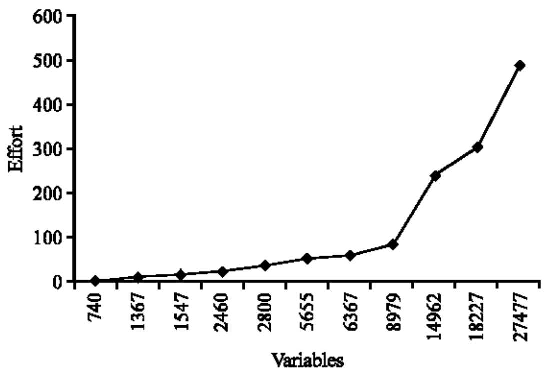


Fig. 4: The relationship between OOPS metric and effort

effort of software, therefore, the required effort of software increase in an exponential diagram by increasing number of classes, attributes and methods in their class diagram (Fig. 4).

## CONCLUSION

Based on previous studies, projects usually last longer than the scheduled time. So, there is no doubt that the correct estimation of time, cost and effort used for performing projects as well as the related affecting factors is an important issue. Assessment of the project size is the first step to estimate the effort of software. Size parameter is assumed as the initial input in most cost estimation models including COCOMO|| Model. Changing estimations may lead to the increase or decrease in the proposed budget of the project. The proposed scheme uses the OOPS metric to decrease the errors of source code size estimations. A parametric study on 12 Java projects indicates that COCOMO|| Model not only provides the software development team with a suitable estimation of the required effort for the project but also provides more optimum values than the previous works in which POPS metric was used to estimate the source code size (Jain and Singh, 2014a, b). Also, there will be a better estimation via the proposed method than the COCOMO|| Model where 15 extra cost drivers have been considered (Garg *et al*., 2014) because in this model, apart from making use of the adjustment factor in estimating effort, the parameter of size is improved too, resulting in the improvement of the project's cost and time estimation. The model also decreases the MMRE compared to the other works. MMRE is calculated equal to 0.1951 in a proposed scheme of COCOMO|| Model in this study and 0.3706 by Jain and Singh (2014a, b). Also, Garg *et al.* (2014) MMRE for 20 evaluated projects is 0.2641 and by Soleimanian *et al*. (2015) which uses a hybrid of Genetic algorithm with a Tabu search algorithm to calculate effort estimates in COCOMO|| Model, MMRE is equal to 0.2973. Also, Soleimanian *et al*. (2015) which use a continuous genetic algorithm, MMRE is equal to 0.2153.

## REFERENCES

Antoniol, G., C. Lokan, G. Caldiera and R. Fiutem, 1999. A function point-like measure for object-oriented software. Empirical Software Eng., 4: 263-287.

Azam, F., S. Qadri, S. Ahmad, K. Khan and A.B. Siddique *et al*., 2014. Framework of software cost estimation by using object orientated design approach. Intl. J. Sci. Technol. Res., 3: 97-100.

Boehm, B.W., 2000. Software Cost Estimation with COCOMO II. Prentice Hall, Upper Saddle River, New Jersey, ISBN:9780130266927, Pages: 502.

Ceke, D. and B. Milasinovic, 2015. Early effort estimation in web application development. J. Syst. Software, 103: 219-237.

Chalotra, S., S.K. Sehra, Y.S. Brar and N. Kaur, 2015. Tuning of COCOMO model parameters by using bee colony optimization. Indian J. Sci. Technol., Vol. 8, 10.17485/ijst/2015/v8i14/70010.

Chen, Y., B.W. Boehm, R. Madachy and R. Valerdi, 2004. An empirical study of EServices product UML sizing metrics. Proceedings of the International Symposium on Empirical Software Engineering ISESE'04, August 20, 2004, IEEE, Redondo Beach, California, ISBN:0-7695-2165-7, pp: 199-206.

Chu, X., 2016. Improving estimation accuracy using better similarity distance in analogy-based software cost estimation. Master Thesis, Uppsala University, Uppsala, Sweden.

Dio, W.L., L.F. Capretz, A.B. Nassif and D. HO, 2015. A hybrid intelligent model for software cost estimation. J. Comput. Sci., 9: 1506-1513.

Garg, K., P. Kaur, S. Kapoor and S. Narula, 2014. Enhancement in COCOMO model using function point analysis to increase effort estimation. Intl. J. Comput. Sci. Mob. Comput., 3: 565-572.

Jain, S. and R. Singh, 2014a. Predictive Object Point metrics (POP): A better size estimator for OO software. Intl. J. Adv. Software Eng. Res. Method., 1: 59-62.

Jain, S., V. Yadav and R. Singh, 2014b. An approach for OO software size estimation using predictive object point metrics. Proceedings of the International Conference on Computing for Sustainable Global Development (INDIACom), March 5-7, 2014, IEEE, New Delhi, India, ISBN:978-93-80544-10-6, pp: 421-424.

Jain, S., V. Yadav and R. Singh, 2014. A simplified formulation of Predictive Object Points (POP) sizing metric for OO measurement. Proceedings of the IEEE International Conference on Advance Computing (IACC), February 21-22, 2014, IEEE, Gurgaon, India, ISBN:978-1-4799-2573-5, pp: 1367-1372.

Johnson, K., 1998. Software cost estimation: Metrics and models. Master Thesis, University of Calgary, Calgary, Alberta.

Jorgensen, M. and D.I. Sjoberg, 2004. The impact of customer expectation on software development effort estimates. Intl. J. Project Manage., 22: 317-325.

Jorgensen, M. and M. Shepperd, 2007. A systematic review of software development cost estimation studies. IEEE Trans. Software Eng., 33: 33-53.

Leung, H. and Z. Fan, 2002. Software Cost Estimation. In: Handbook of Software Engineering and Knowledge Engineering, Chang, S.K. (Ed.). World Scientific, Singapore, ISBN:981-02-4514-9, pp: 1-14.

Mittas, N., I. Mamalikidis and L. Angelis, 2015. A framework for comparing multiple cost estimation methods using an automated visualization toolkit. Inf. Software Technol., 57: 310-328.

Mittas, N., M. Athanasiades and L. Angelis, 2008. Improving analogy-based software cost estimation by a resampling method. Inf. Software Technol., 50: 221-230.

Pfleeger, S.L., F. Wu and R. Lewis, 2005. Software Cost Estimation and Sizing Methods: Issues and Guidelines. RAND Publisher, Santa Monica, California, ISBN:0-8330-3713-7,.

Saez, A.M.F., M. Genero, D. Caivano and M.R. Chaudron, 2016. Does the level of detail of UML diagrams affect the maintainability of source code?: A family of experiments. Empirical Software Eng., 21: 212-259.

Soleimanian, F., F.S. Gharehchopogh and A. Pourali, 2015. A new approach based on continuous genetic algorithm in software cost estimation. J. Sci. Res. Dev., 2: 87-94.

Xu, Z. and T.M. Khoshgoftaar, 2004. Identification of fuzzy models of software cost estimation. Fuzzy Sets Syst., 145: 141-163.

Zhou, Y., Y. Yang, B. Xu, H. Leung and X. Zhou, 2014. Source code size estimation approaches for object-oriented systems from UML class diagrams: A comparative study. Inf. Software Technol., 56: 220-237.