ISSN: 2788-7243 ISSN: 2788-7235

# Non-Algorithmic Software Effort Estimation - Challenges and Solutions

*Rasha Assaf Dr. Fasil Khamayseh jomana Khaseep

## *Abstract*

software effort estimation is the first step in every software development process. The accuracy of the software effort estimation can seriously affect the success of projects. Over estimating will result in allocating excess and unneeded resources that will cost the company unexpected expenses, while under estimation will lead to delays in software delivery. The dynamic and evolving nature of the software development process makes it challenging to build a reliable effort estimation system. One of the common models used for an efficient software cost estimation is the Constructive Cost Model (COCOMO) model. The COCOMO model is flexible, it is used in different environments. In this paper we aim to explore non-algorithmic approaches to effort estimation that attempts to improve upon the COCOMO model. We will focus on machine learning based methods trained on a public dataset. Furthermore, we will present some of the datasets available to use in the software development process and how they can be utilized to train machine learning models.

*Palestine Polytechnic University

ISSN: 2788-7243 ISSN: 2788-7235

## INTRODUCTION

Software cost estimation is one of the most important activities in software project management, but it remains a challenging and difficult task. This is the case since software development is a dynamic and constantly changing process. Also, software applications that have developed in size and complexity, and the requirement for effort estimation have increased in return. Software development specialists and researchers have been trying to develop methods to estimate software costs and schedules [1]. Software cost estimation models have appeared in the literature over the past four decades. Nonetheless, the domain of software cost estimation is still in its earliest stages. Project managers are responsible for planning the software development life cycle, allocate resources and estimate cost. Poor project effort estimation leads to project failures and pernicious finding for the project. Additionally, documentation is a key part of the development process that should also be accounted for. The product should be released with instructions and manuals [2]. One should also account for the software maintenance, which continues for the life of the software. This phase may include bug fixes and new releases. The Software development Life Cycle shown in Figure 1.

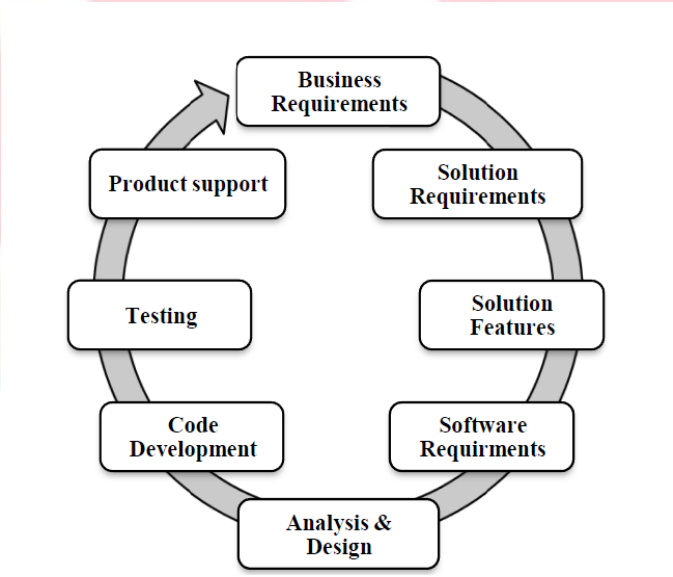


**Fig 1: Software Development Life Cycle [2]**

Software cost estimation techniques can be categorized into two types which are algorithmic and non-algorithmic models.

ISSN: 2788-7243 ISSN: 2788-7235

Algorithmic models depend on the statistical analysis of historical data [3]. Some of the well-known algorithmic models are Albrecht's Function Point [4, 5], Software Life Cycle Management (SLIM) and the Constructive Cost Model (COCOMO) [6].

Non-algorithmic methods depend on new approaches such as Parkinson [6], Expert Judgment, Price-to-Win and machine learning methodologies like regression trees, rule induction, fuzzy systems, genetic algorithms, Bayesian networks, artificial neural networks, and evolutionary computation.

This paper is structured into six sections as follows: Section II provides the motivation and existing work on COCOMO modeling. Section III describes the COCOMO models. Section IV covers the effort estimation techniques. Section V describes the evaluation metrics for the machine learning based approaches. Section VI provides insights on the existing work, challenges and possible solutions and we conclude in section VII.

## MOTIVATION AND PREVIOUS WORK

Effort estimate for software development has not been reliable and in many cases the available techniques used today by companies and managers can lead to under or over estimation. Under estimation is when the project costs more money, more man hours and can lead to delays in delivering the software. Over estimation is when we allocate much more resources than needed, costing the company more money.

There is no accurate cost/schedule estimation model that one can rely on to perform effort estimation due to the dynamic nature of software development and its complexity. Many techniques used for the effort estimation are algorithmic, but recently researchers started exploring non-algorithmic methods that are data driven such as machine learning. Machine learning based methods can provide different perspectives and have shown to outperform other traditional approaches.

Software effort estimation methods can improve the planning of the software development projects. Having the ability ro project the cost of software development is of prime significance. Researchers have recently started considering a set of approaches that are based on delicate computing methods. These techniques include artificial neural networks, fuzzy logic models and genetic algorithms. Artificial neural networks are able to generalize from trained data sets. Over a known set of training data, a neural-network learning algorithm develops rules that fit the data and predicts previously unseen data

ISSN: 2788-7243 ISSN: 2788-7235

in a reasonable pattern [11-13].

Chen et. al. [10], proposed a model based on feature subset selection to find a subset that gives similar, if not superior, performance than using all the attributes by removing redundant ones. They use the WRAPPER FSS method [10]. The outcome of this study shows the important improvement for mean prediction values without any increase for variance.

Idri et. al. [11], proposed a model based on Fuzzy sets instead of the traditional COCOMO' 81 model. They defined identical fuzzy sets for each cost driver and its related linguistic values. They represented these fuzzy sets by trapezoid shaped membership functions. The outcome of these ways is affected by accuracy in a positive way [11].

Saljoughinejad et. al. [13], they analyzed the cost drivers utilize meta-heuristic algorithms. The improvement of these methods of COCOMO is clearly done by effective selection of coefficients and reconstruction of COCOMO. Three meta-heuristic optimization algorithms are carried out synthetically to enhance the method of COCOMO model. The findings of the proposed method are compared to COCOMO itself and other existing models. This comparison unequivocally uncovers the prevalence of the proposed strategy [13].

## COCOMO MODELS

COCOMO was first published in 1981 Barry W. Boehm's Book Software engineering economics [1] as a model for estimating effort, cost, and schedule for software projects. It drew on a study of 63 projects at TRW Aerospace where Barry Boehm was Director of Software Research and Technology in 1981. These projects were based on the waterfall model of software development which was the predominant software development process in 1981. This model was presented utilizing constant parameters as well as applying statistical data regression analysis based on 63 various software projects.

References to this model typically call it COCOMO 81. In 1997 COCOMO II was developed and finally published in 2000 in the book Software Cost Estimation with COCOMO II [2]. COCOMO II is the successor of COCOMO 81 and is better suited for estimating modern software development projects. It provides more support for modern software development processes and an updated project database. The need for the new model came as software

ISSN: 2788-7243 ISSN: 2788-7235

development technology moved from mainframe and overnight batch processing to desktop development, code reusability and the use of off-the-shelf software components. This article refers to COCOMO 81 [6].

The Constructive Cost Model (COCOMO) is an algorithmic software cost estimation model developed by Barry Boehm. The model uses a basic regression formula, with parameters that are derived from historical project data and current project characteristics. There are two releases of the COMCOM Model: COCOMO 81 and COCOMO II.

The COCOMO models 81 and II , the effort needed to develop a project can be written inside the following standard form (1) [8]:

$$PM = \alpha * (KLOC)^b * \left(\prod_j EM_j\right) \quad (1)$$

where

$PM$ = effort estimate in person months

$\alpha$ = productivity coefficient

$b$ = economies (or diseconomies) of scale coefficient.

$KLOC$= kilo lines of code

$EM$= effort multipliers or cost drivers

TABLE I: COCOMO 81 vs COCOMO II

| BASIS FOR COMPARISON | COCOMO 1 | COCOMO II |
|---|---|---|
| Basic | Founded on linear reuse formula | Based on the non-linear reuse formula. |
| Size of software stated in terms of | Lines of code | Object points, function points and lines of code |
| Number of sub models | 3 | 4 |
| Cost drivers | 15 cost drivers as shown in table 2 with rating scale of sex of linguistic values | 17 |
| Model framework | Development begins with the requirements assigned to the software. | It follows a spiral type of development. |

ISSN: 2788-7243 ISSN: 2788-7235

| | | |
|---|---|---|
| Data points | 63 projects referred | 161 projects referred |
| Estimation precision | Offers estimates of effort and schedule. | Supplies estimates that represent one standard derivation nearly the most likely estimate. |

TABLE II: the 75 effort multipliers used in intermediate model cost drivers[6]

| | Ratings | | | | | |
|---|---|---|---|---|---|---|
| Cost Drivers | Very Low | Low | Nominal | High | Very High | Extra High |
| **Product attributes** | | | | | | |
| Required software reliability | 0.75 | 0.88 | 1.00 | 1.15 | 1.40 | |
| Size of application database | | 0.94 | 1.00 | 1.08 | 1.16 | |
| Complexity of the product | 0.70 | 0.85 | 1.00 | 1.15 | 1.30 | 1.65 |
| **Hardware attributes** | | | | | | |
| Run-time performance constraints | | | 1.00 | 1.11 | 1.30 | 1.66 |
| Memory constraints | | | 1.00 | 1.06 | 1.21 | 1.56 |
| Volatility of the virtual machine environment | | 0.87 | 1.00 | 1.15 | 1.30 | |
| Required turnabout time | | 0.87 | 1.00 | 1.07 | 1.15 | |
| **Personnel attributes** | | | | | | |
| Analyst capability | 1.46 | 1.19 | 1.00 | 0.86 | 0.71 | |
| Applications experience | 1.29 | 1.13 | 1.00 | 0.91 | 0.82 | |
| Software engineer capability | 1.42 | 1.17 | 1.00 | 0.86 | 0.70 | |
| Virtual machine experience | 1.21 | 1.10 | 1.00 | 0.90 | | |
| Programming language experience | 1.14 | 1.07 | 1.00 | 0.95 | | |
| **Project attributes** | | | | | | |
| Application of software engineering methods | 1.24 | 1.10 | 1.00 | 0.91 | 0.82 | |
| Use of software tools | 1.24 | 1.10 | 1.00 | 0.91 | 0.83 | |
| Required development schedule | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 | |

## COST ESTIMATION TECHNIQUES

There are many modern models available for estimating the effort and schedule of the software development project.

The methods are categorized into two types: Algorithmic and Non-algorithmic. Some of the popular estimation methods are shown in Figure 2.

ISSN: 2788-7243 ISSN: 2788-7235

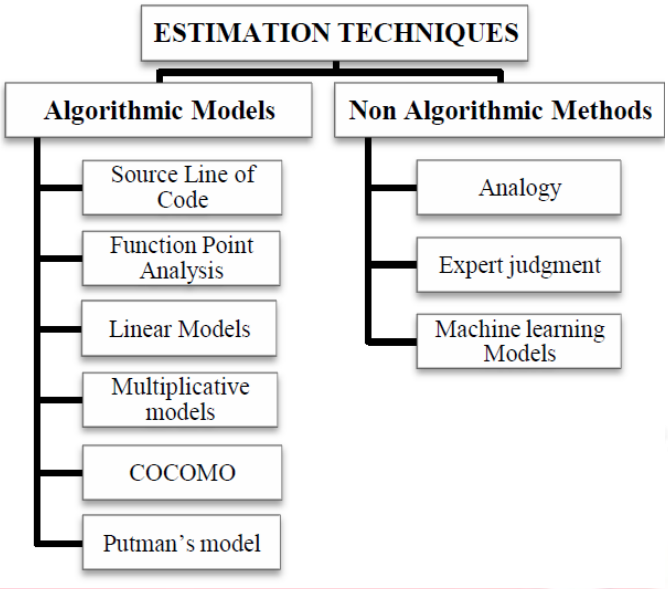


**Fig 2: Estimation Techniques [2]**

In this paper we will focus on non-algorithmic solutions such as machine learning. There are many machine learning methods that can be used for the COCOMO project. Using machine learning, the goal is to learn a function that can estimate the effort and cost in software development. The objective of the machine learning model is to minimize the mean square error or the Mean Magnitude of Relative Error (MMRE). There are ways to train such models.

The authors in [14] used the COCOMO dataset, which contains 20 features and has one target/label, the effort to be estimated. The dataset is very small, it contains 63 examples for training and 15 examples for testing. The 63 examples were used to train a neural network model. Their best performing model contained two hidden layers with 18 and 15 neurons in each layers, respectively. This model gave the lowest deviation from the actual effort. That being said, the dataset is very small to train a reliable neural network.

There are other ways to estimate effort that can rely on natural language processing (NLP). For instance, the Kaggle dataset [15] contains about 23,000 tasks described in natural language and each task has corresponding effort or points. With the advancements in seq2seq modeling and natural language processing techniques, one can train a model that estimates effort based on the language of the task. For instance, we can use LSTM based model or transformer model (i.e. BERT) to train effort estimation model. The architecture of the model contains the encoder (LSTM based or BERT), which maps the text found in the task to a representation (embedding), followed by a dense layer and softmax layer that generates the output label.

ISSN: 2788-7243 ISSN: 2788-7235

**Evaluation Methods**

COCOMO'S performance measure by calculated relative error, which is the difference between actual and estimated effort relative to the actual effort. As in equation (2):

$$Relative\ Error\ (RE) = \frac{|Estimated\ effort - Actual\ effort|}{Actual\ effort} \quad (2)$$

The evaluation will compare the accuracy of the estimated effort with the actual effort. Most researchers applied the most recurrent, one which is Magnitude of Relative Error (MRE) which is defined as in equation (2):
Mean Magnitude of Relative Error (MMRE) [9] is used to validate errors of estimation based on Magnitude of Relative Error (MRE), both of which are defined below:

$$MRE = \left| \frac{effort_{actual} - effort_{estimated}}{effort_{actual}} \right|$$

$$MMRE = \frac{1}{N} \sum_{i=1}^{N} MRE_i$$

- Prediction level: *PRED(I) = K/N*
  - *K is the number of estimation with MRE<=1*
  - *Commonly used PRED(0.30) and PRED(0.25)*

**CHALLENGES TO EFFORT ESTIMATION**

One of the challenges for effort estimation is lack of data. We have seen researchers that attempt to apply complex machine learning models with very small datasets. For instance, in [14], the authors used 63 examples to train a neural network and 15 examples for testing. This is a very small dataset to train a neural network and one needs a bigger dataset to learn neural network parameters. With this dataset size, it is possible that simpler methods such as decision trees may perform better than neural networks.

ISSN: 2788-7243 ISSN: 2788-7235

That being said, a dataset of such size is very small for any machine learning method to learn a reliable function that maps inputs to the output.

The best solution is generate more data, if the data is rare and not available, then one has to turn into data augmentation techniques, up-sampling of data or simulate the data to increase its size. The other issue to look at is how well the data is balanced and if the models are overfitting or under_fitting the data. In some cases like that, we can use ensemble learning methods such as bagging and boosting. Bagging will learn multiple weak classifiers and then use majority voting to predict the label. Boosting also learns multiple classifiers, but it will assign higher weights to the examples that are harder to learn and it will also assign a weight to the classifiers that have better performance. This can be beneficial for small datasets.

## CONCLUSION

Software effort estimation is a challenge, yet it is important for making high-quality management decisions in the software industry. to be noted that the most important reason for the software project failure is inaccurate estimation of parameters in early stages of the project planning. In this paper, we described the popular COCOMO model for software effort estimation. We also described the two possible approaches to effort estimation, which fall under two categories: algorithmic and non-algorithmic. Machine learning based approaches fall into the non-algorithmic category, they are data driven and data hungry. We have observed that many of those machine learning methods are being applied on small datasets. Some of the common techniques used include fuzzy logic and neural networks. We also pointed out that using a small dataset to train a neural network may not be the best option and ensemble learners might perform better.

## REFERENCES

Barry, B. (1981). Software engineering economics. New York, 197.

Boehm, B. W., Madachy, R., & Steece, B. (2000). Software cost estimation with Cocomo II with Cdrom. Prentice Hall PTR.

Boehm, B. W., Madachy, R., & Steece, B. (2000). Software cost estimation with Cocomo II with Cdrom. Prentice Hall PTR.

ISSN: 2788-7243 ISSN: 2788-7235

Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., & Selby, R. (1995). Cost models for future software life cycle processes: COCOMO 2.0. Annals of software engineering, 1(1), 57-94.

Chandrasekaran, R., & Kumar, R. V. (2012). On the Estimation of the Software Effort and Schedule using Constructive Cost Model–II and Functional Point Analysis. International Journal of Computer Applications, 44(9), 38-44.

Chen, Z., Menzies, T., Port, D., & Boehm, B. (2005, May). Feature subset selection can improve software cost estimation accuracy. In Proceedings of the 2005 workshop on Predictor models in software engineering (pp. 1-6).

Goyal, S., & Parashar, A. N. U. B. H. A. (2018). Machine learning application to improve COCOMO model using neural networks. International Journal of Information Technology and Computer Science (IJITCS), 3, 35-51.

Goyal, S., & Parashar, A. N. U. B. H. A. (2018). Machine learning application to improve COCOMO model using neural networks. International Journal of Information Technology and Computer Science (IJITCS), 3, 35-51.

Hodgkinson, A. C., & Garratt, P. W. (1999). A neurofuzzy cost estimator. In Proceedings of the Third Conference on Software Engineering and Applications (pp. 401-406).

Idri, A., Abran, A., & Kjiri, L. (2000, March). COCOMO cost model using fuzzy logic. In 7th international conference on fuzzy theory & techniques (Vol. 27).

Khatibi, V., & Jawawi, D. N. (2011). Software cost estimation methods: A review

Madheswaran, M., & Sivakumar, D. (2014, July). Enhancement of prediction accuracy in COCOMO model for software project using neural network. In Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT) (pp. 1-5). IEEE.

ISSN: 2788-7243 ISSN: 2788-7235

Saljoughinejad, R., & Khatibi, V. (2018). A new optimized hybrid model based On COCOMO to increase the accuracy of software cost estimation. Journal of Advances in Computer Engineering and Technology, 4(1), 27-40.

Zia, Z., Rashid, A., & uz Zaman, K. (2011). Software cost estimation for component-based fourth-generation-language software applications. IET software, 5(1), 103-110.