Journal of Software Engineering and Applications, 2019, 12, 226-248
http://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

A Survey on Software Cost Estimation Techniques

Sai Mohan Reddy Chirra, Hassan Reza

School of Electrical Engineering & Computer Science, University of North Dakota, Grand Forks, USA
Email: saimohanreddy.chirra@und.edu, hassan.reza@engr.und.edu

How to cite this paper: Chirra, S.M.R. and Reza, H. (2019) A Survey on Software Cost Estimation Techniques. *Journal of Software Engineering and Applications*, **12**, 226-248.
https://doi.org/10.4236/jsea.2019.126014

Received: January 26, 2019
Accepted: June 27, 2019
Published: June 30, 2019

Abstract

The ability to accurately estimate the cost needed to complete a specific project has been a challenge over the past decades. For a successful software project, accurate prediction of the cost, time and effort is a very much essential task. This paper presents a systematic review of different models used for software cost estimation which includes algorithmic methods, non-algorithmic methods and learning-oriented methods. The models considered in this review include both the traditional and the recent approaches for software cost estimation. The main objective of this paper is to provide an overview of software cost estimation models and summarize their strengths, weakness, accuracy, amount of data needed, and validation techniques used. Our findings show, in general, neural network based models outperforms other cost estimation techniques. However, no one technique fits every problem and we recommend practitioners to search for the model that best fit their needs.

Keywords

Software Cost Estimation, Classical SCE Models, Algorithmic Models, Non-Algorithmic Models, Learning-Oriented Cost Estimation Techniques

1. Introduction

Software cost estimation is one of the crucial activities of the software development which involves predicting the effort, size and cost required to develop a software system or a software project [1] [2]. Several cost estimation models have been developed to better estimate the cost of a project. To have better accuracy in software cost estimation it is important to consider appropriate approaches to apply. Inaccurate effort estimations have been found to be very risky in the field of industrial economics. Over the past few decades, conducting cost estimation for different projects was cumbersome, but with the implementation of the software cost estimation process, things have positively changed. As it will be highlighted in this paper, there are different types of software cost estimation models which can be categorized into Algorithmic models, non-algorithmic models and learning oriented models [3]. The Algorithmic models include COCOMO, Function point analysis, Putnam model, etc. The non-algorithmic models which are also called as the non-parametric models include Expert judgment, analogy based, price to win, top-down and bottom up. The learning-oriented models or the machine learning methods include Artificial Neural Networks (ANN's), Fuzzy Logic (FL), analogy based, Bayesian Network, Regression tree, Support Vector Machines, Genetic Algorithm (GA) and case-based reasoning [1]. The different methods have their pros and cons which imply that there are not the same in the sense of performance. The discussion will give more credential to modern methods of machine learning and their impacts on the software engineering sector.

Software Cost Estimation as a topic entails several issues as it requires keen observation and frequent trials before stipulating that a certain technique is fit for the estimation purposes. The utilization of software cost estimation techniques makes it possible to predict the amount of effort and cost that will be incurred in a certain software project. Thus, the approximated amount of manpower needed, and the period required to complete the project in the required time are all made possible by the software cost estimation process (**Figure 1**).

However, it still remains to be one of the most difficult areas of software engineering as each achievement made in the area attracts more questions and research. The algorithmic models primarily rely on the mathematical formulas and expressions to give a prediction on a project. The formulas exploited in the sector are also dependent on different factors such as product factor to calculate the estimations [3]. The non-algorithmic methods are considered to be most advanced as they have incorporated artificial intelligence in achieving their results. The field of artificial intelligence (AI) is a different scientific spectrum that integrates automation with non-algorithmic methods to come up with more accurate results [1]. The current methods have been improvised through artificial intelligence such that they can easily estimate the cost of a project even where limited data is available. More automation and improvements have been performed on the process which has led to the emergence of more strong methods.

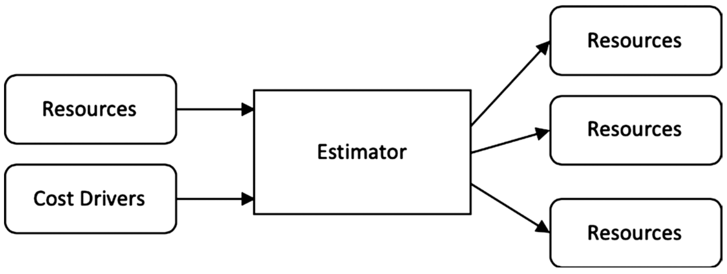

Figure 1. Cost estimation process [3].

Software cost estimation has the potential to completely change the industry by providing an accurate prediction of the amount of resources a project might need to be completed. However, at the moment, these estimation techniques could lead to important negative effects. The capability to estimate the cost and effort a project will take is of great importance because overestimation can easily lead to incurring of financial losses in any organization [1]. Under-estimation on the other hand, can significantly contribute to poor quality service delivery leading to failure of the entire project [1]. In a study, it has been reported that there is an overestimation of up to 40% in the estimating of software projects [4]. There is a need for efficient and accurate estimations to reduce the risks and timely delivery of a software project within the budget constraint [1]. This paper will address the strengths and weakness of different cost estimation techniques used in software cost estimation. Through the discussion, it is plausible for the researchers and the practitioners to understand the correct area where each model could be deployed and the factors that make it best suited for the specific area. All these issues will be highlighted and discussed in this paper (**Figure 2**).

2. Background and Related Work

Several studies and systematic reviews related to the software cost estimation to techniques have been conducted to date [1]-[10]. The SLR papers gave the timeline of the cost estimation methods while the studies conducted gave a discussion on the existing cost estimation methods. There is still a research gap in listing out the popular cost estimation techniques and discussing the strength and weakness of those cost estimation techniques which aids the researchers and practitioners to choose which estimation technique to opt for depending on their needs. According to Jianfeng *et al.*, software development effort estimation (SDEE) is a process that is used by the project managers or the software developers in predicting the effort required to develop a software system [2]. Over the

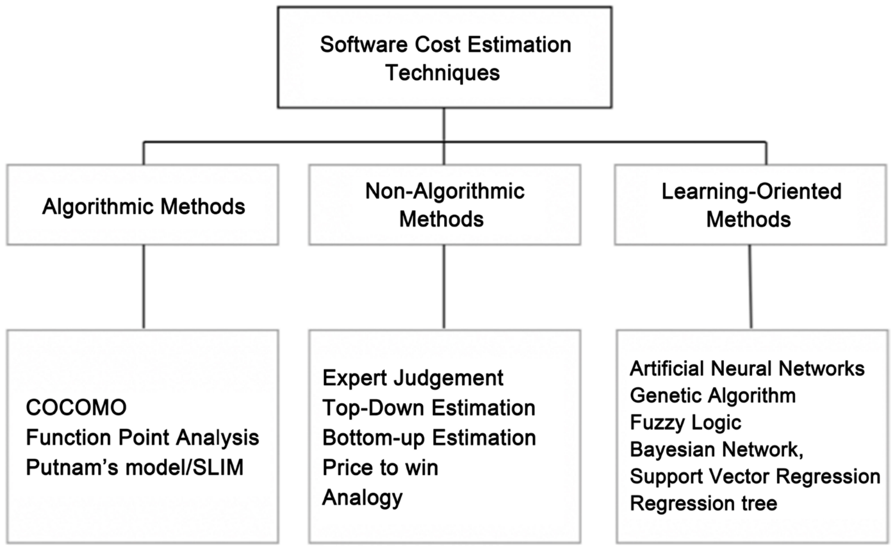

Figure 2. Software cost estimation techniques [7].

years since the 1990's, researchers have suggested the implementation of Machine Learning (ML) models and SDEE in improving on estimation accuracy. Although there have been numerous researches on the process still there lacks empirical evidence of comparisons of different software cost estimation techniques. There are ongoing studies and researches that are underway to give detailed literature on the issue of software cost estimations as depicted by Sharma *et al.* [1]. The current researchers are aimed at improving o the functionality of the different cost estimation approaches that are used in software cost estimation. According to Jorgensen *et al.*, it is of great relevance for the users of these approaches to be well conversant with each approach so as to aid in selecting the appropriate technique to deploy in software cost estimation [5].

Current studies show that in order for us to be able to understand the achievements that have been met since the software cost estimation process began it would be of importance to review the methods back to decades when they were exploited [3]. Since then to the current year, companies have been exploiting different taxonomies and classification criteria in identifying the suitable method to support their estimations [7]. The first journals and reports were published between 50's and 70's which proves that studies on software cost estimation existed in the past despite them being conducted manually [8]. With numerous studies being perfumed on the process we are aware that the entire process entails software plans, resources, coding, testing, development and design. Numerous organizations are dependent upon software development for their stability and sustainability in relation to cost estimations. According to [8], it is important for the cost of software to be estimated in such a way that it will not compromise quality, efficiency and timeline. The former methods that were exploited in the sector were dependent on source line of code (SLOC), cost drivers and function points [3].

However current studies prove that there is a lot of automation in the software cost estimation process with each technique being implemented to fit the desired functions [1]. There are basic reasons as to why there is a tremendous research going on in software cost estimation process; proper budgeting, accurate estimations, software improvement investment analysis, project planning and control, trade-off and risk analysis.

3. Approaches

3.1. Algorithmic Methods

1) *COCOMO* (*Cost Constructive Model*)

COCOMO is one of the classical techniques used in cost estimation [11]. There are different approaches that can be undertaken when it comes to estimating the cost of software projects. Among these methods that are available, COCOMO (Constructive Cost Model) is the most common model that is majorly used [11]. It was developed by Barry Bohem in 1981 [11]. The COCOMO falls under the algorithmic technique. The model has been in existence for many years and since it is being up to date serves to indicate that it is highly reliable as a cost estimation technique.

COCOMO is a software cost estimation approach that uses mathematical formulas and calculations to estimate the cost of a project. It gives the estimates regarding the amount of the effort that is required as well as the schedule for the software project [11]. Thus, it can achieve the two most vital objectives of cost estimation which ascertain the cost of the essential resources and schedule for the software project. In addition to the above, the Constructive Cost Model uses a set of metrics that guide its operations. Function Points (FP) and Object Points (OP) are the metrics that guide the calculations, and at the same time, they endure that the calculations are in line with codes relating to KLOC and KDSI [11]. COCOMO II, as well as COCOMO 81, are the two versions of Constructive Cost Model. The model parameters in the Cost Constructive model are derived from fitting a regression formula using historical projects data [11]. A total of 61 projects are used for COCOMO 81 and 163 projects for COCOMO II [11].

2) *COCOMO* (*Cost Constructive Model*) 81

To being with, COCOMO 81 was the first version of this technique. Under this technique, estimates that are produced, according to [11], have a 20% accuracy margin for the actual value of the software while 68% is the accuracy margin for the time estimate. In the same COCOMO model, there are three sub-models which apply throughout the lifecycle of the project. They include; the basic model, the intermediate model, and advanced model. The basic model is applicable in early life of a project. It is essential in providing a rough estimate of what is to be expected in the later stages of the project. It offers a glimpse into what is to be expected and how activities need to be undertaken to meet a set of requirements [11]. The intermediate model, on the other hand, is different in its way in that it deals with the estimation of value and time after more details about the project have been acquired. With the detailed requirements, this model can effectively initiate the cost estimation process. The advanced model comes as the last applicable model with COCOMO 81. It only applies upon completion of the project. It offers a more refined estimate that is useful and reliable.

There are different equations that are used with COCOMO 81. The two equations that are used with COCOMO help in calculating the effort and scheduled time. The estimated schedule time is measured in months. The two equations [11] include;

$$\mathrm{PM} = \mathrm{a}\left(\mathrm{KDSI}\right) * \mathrm{EAF} \tag{1}$$

$$\mathrm{TDEV} = \mathrm{c}\left(\mathrm{PM}\right)\mathrm{d} \tag{2}$$

Each abbreviation in the equations stands for a factor that affects the cost of software. Thus, they present dynamics that help in cost estimation.

PM => Person-Months

EAF => Effort Adjustment Factor

TDEV => Scheduled time

KDSI => Number of lines of code (that is denoted in thousands)

Initials a, b, c and d are constants which result from the mode that is used in estimating cost.

The modes include organic, semi-embedded, as well as embedded. On the same note, there are cost drivers that related to COCOMO 81 [11]. For instance, the EAF serves the purpose of tailoring the estimate so that conditions that affect the development of the environment are considered. When it comes to the intermediate model, a total of 15 drivers are present that affect cost which can be manipulated into helping to calculate for the EAF [11]. The 15 cost drivers are segmented into four sections which are product attributes, personal attributes, computer attributes and project attributes [11]. Depending on the impact that the cost drivers have on the project development, they are categorized ranging from very low to extra high [11].

On a different note, various advantages and disadvantages are associated with the use of COCOMO 81. Its major advantage is that it is simple to estimate cost with this technique [11]. It is useful to estimate in large projects and takes less time to estimate the cost. On the other hand, there are various disadvantages that are associated with the use of this model. With the Constructive Cost Model, there is a disadvantage of estimation failures. Estimation failures come about in that estimation is carried out at the early stages of the project. This leaves room for many errors, and most of these errors can result in failures in estimation. At the early stages of the project many variables have not evolved, and thus they cannot be considered in the estimation process. All the same, the early stages do not offer sufficient ground and room for cost estimation that is reliable. In doing so, this form of cost estimation is not reliable and cannot meet the needs of the current software dynamics.

3) *COCOMO* (*Cost Constructive Model*) *II*

COCOMO II is different from COCOMO 81 considering that is addresses most of the problems and challenges that were associated with COCOMO 81 is its inability to offer reliable estimates due to many failures [12]. Additionally, the inability of COCOMO 81 to gather all input parameters relating to size and time was another problem. However, COCOMO II addresses these issues considering that it does not base its estimates at the earlier stage of the project [12]. In addition to the above, COCOMO II also aims to develop database for software cost estimates that are characterized with tool capabilities meaning that it can enhance and ensure there is the model improvement. Thirdly, and lastly, the model also aims to allow the provision of the quantitative and analytical framework that enhances the evaluation of effects that result from software technology improvements. This distinct model is not very traditional considering that it allows for the incorporation of technological changes when it comes to the evaluation process. Thus, it is effective in that makes it possible to carry out an evaluation based on additions made to a software project.

All the same, [12] explains that this approach is not very different from

COCOMO 81. In other terms, the minor changes that are notable in this model are the use of a higher number of cost drivers. The cost drivers that are used are different as compared to those that were used in COCOMO 81. When it comes to calculating to come up with the estimates, the different approach that is undertaken is that variables are used instead of constants. In addition to the above, lines of code are used as the main metric as opposed to function points that are used as the main metric in COCOMO 81 [12]. However, function points can also be used in place of lines of code for making estimates. In this case, the line of code metric tools is customized to act as the LOC. The three models that are characteristic of COCOMO II are Application Composition Model, Early Design Model, and Post-Architecture Model. The Application Composition Model is used for projects that have been built for rapid application [12]. Object points are useful when it comes to ascertaining size estimates. Prototyping efforts are employed to help in resolving high-risk issues. Thus, it easily meets the expectations and the needs of many users making it one of the most used traditional cots estimation technique favorable for cost estimation of software being currently developed.

Various advantages and disadvantages are associated with COCOMO II. One of its advantages is that COCOMO II has a calibration process that is clear and effective. It is clear and effective in that it has systems in place which clearly define the main metrics and the variables that are used are more detailed. In addition to the above, there is the advantage of allowing industries to function more effectively due to their ability to adopt a model that is flexible to changes. Thus, COCOMO II is an industry model.

3.1.1. Function Point Analysis

Due to diverse functional aspects in software systems, proper metric systems remain to be a major concern in software engineering. Therefore, software engineers focus on measuring the functionality size in software development projects. In reducing the complex task of software metrics in terms of functional size, functional point analysis method was developed [13]. Functional point analysis refers to the standardized methods of determining software sizes by using functional constraints which determine the key features to be designed [14]. Significantly, this method is universal because its application is not limited to programming languages and technologies. In function point analysis, two major components are measured. The most crucial aspects of software application measure in function point analysis comprise the data functionality and transaction functionality attributes [13]. Precisely, the metrics of the two basic features are determined by evaluating the scope of the system product, quality indicators, productivity, and the system performance.

Function point analysis (FPA) evaluates the system's metrics from a functional perspective, thereby resolving issues associated with technology dependency in the development lifecycle [13]. The efficiency of FPA in software engineering is achieved through a comprehensive analysis of applications in three stages [13].

The first stage of function point analysis concerns identifying the forms of transactions to be made in the software applications. Secondly, the engineers evaluate and appraise the components of the software system. Lastly, the process involves evaluations of the general system characteristics. Fundamentally, general system characteristics are categorized using into 14 main features [13]. These are data processing, system performance, hardware configurations, transaction rates, data entry, end-user efficiency, online updates, reusability, ease of use, support of multiples sites and change facilitations [13]. There has also been tremendous research in enhancing function point analysis with non-functional requirements.

3.1.2. Putnam's Model

The Putnam's model is a dynamic multivariate model which was developed by Larry Putnam in the late 1970s for effort estimation [15]. The model functions by examining the many software projects and analyzing the distribution of manpower. The relationship between the size and effort is non-linear and is highly sensitive to delivery time. It provides a simple and computationally plausible way of predicting software costs. It is used to calculate both effort and time required to complete a software project based upon the specified size of the project. The Putnam's model equation is given as [15]:

$$\frac{B^{1/3} * \text{Size}}{\text{Productivity}} = \text{Effort}^{1/3} * \text{Time}^{4/3} \tag{3}$$

With this model, SLIM is the tool that is useful when it comes to cost estimation and allows workforce scheduling. Thus, it achieves the objectives of cost estimation as it provides the schedule and cost of the essential resources. However, it's capacity in estimating the total manpower requirements and development time at an early stage is still not satisfactory [16]. It has the setback of not accounting for other aspects of the software project [16]. There are a series of aspects relating to software development especially in its life cycle that needs to be considered. The uncertainty in the size of the software may lead to the inaccurate cost estimation. According to [17] SLIM's error percentage is said to be 772.87%. On the contrary, its advantage is that the model is based on two variables that are keys to cost estimation which are time and size [16] and it needs fewer parameters compared to COCOMO 81 and COCOMO II.

3.2. Non-Algorithmic Methods

3.2.1. Expert Judgement

Expert judgment is one of the traditional techniques that are used in the software cost estimation in the early phases of the software development [3]. The technique is such that it relies heavily on the expertise and the experience of an expert at cost estimating. It depends on the domain knowledge of the expert rather than the historical data [3]. The experienced estimator is tasked with the responsibility of estimating the cost of software based on the fact that they have sufficient knowledge that ensures cost estimation is as accurate as possible [3]. On the same note, the expert judgement used to estimate cost on a given project is often limited to a specific filed of expertise of the estimator. An expert who has knowledge and experience on a specific field is more likely to have majored in the given project putting them in a better position to estimate the cost. Expert judgement comes in handy especially when there are limitations that limit effective and efficient data collection [18]. In doing so, expert judgement is required to make decisions based on the limitations and the stringent factors. Delphi technique is one of the examples that follow the expert judgement approach. An expert can be able to offer an honest and experienced opinion on the best course of action when it comes to an understanding of the impacts of a system being incorporated. There is also a setback of tedious processes that are undertaken in documenting the factors that an expert requires to make the judgement [3]. Most of the factors in many software projects are many and documenting them is tedious as well as difficult. There is no specific validation for this approach as the estimation depends solely on the domain knowledge and previous experiences of the expert.

Additionally, there is the disadvantage of obtaining cost estimates that are biased and optimistic. Expert judgement is given by experts who have human emotions and is very likely that their emotions influence the judgement process [3] [18]. Therefore, there is the possibility that pessimism, optimism, as well as bias might influence the judgement process.

3.2.2. Top-Down Estimation

Top Down cost estimation approach focuses on estimating the cost of a project from the global properties of the overall project and using either algorithmic such as Putnam model or non-algorithmic methods [19]. The estimation is then split into various components in proportion [19]. This method can be followed when there is limited historical data available about the similar project. This technique is more beneficial while the project is still in its early stages. This is because, at this stage, there is no need for detailed information about the project [19].

Top Down estimation is used for high-level decisions when the planning horizon is quite long. This technique is used when there is very little specific project information where we can get a ballpark estimate. Unlike other cost estimation techniques, this approach focuses on activities like management and integration which are overlooked in other techniques. The major disadvantage is that the ballpark estimate is very inaccurate [19].

3.2.3. Bottom-Up Estimation

This is the exact opposite of the top-down estimation methodology. In this particular technique, the cost of every component of the software is derived and then the final result is obtained by combining these elements to get the total estimated cost of the project [19]. The aim of this technique is to obtain a proper estimate that will be an accumulation of the estimates of the smaller components of the software. Depending upon the variety of the projects both the methodologies are useful [19]. The estimation methodology is best suited for small projects to estimate the cost.

In the bottom-up estimation, we take each work package or each activity in a schedule and assign a dollar cost to that and then add them up to get the overall cost of the project [19]. This can create a very detailed estimate that will be very accurate. However, there are some disadvantages of the bottom-up estimate.

1) If we add up each activity, we may be lacking any coordination between the activities such as resource overlap or if some of the dependencies that one activity has to be finished before another can start.

2) The accuracy is high, but it can also take a lot of time to create this level of estimate, which means that it can be expensive to create.

So, to overcome these disadvantages it is preferable to do a high-level estimate *i.e.* top-down estimate for the whole project and then do some detailed estimates for the activities that are coming up in the short term.

3.2.4. Price-to-Win Estimation

In this approach, the estimation of the software project is directly proportional to the budget of the customer. The customer's budget is more focused rather than the functionality of the software and the project costs whatever the customer has to spend on it [19]. This approach is not so recommended as instead of focusing on software functionality it focuses more on the client's budget and capacity. Accuracy varies drastically based on the client's budget and therefore it is rated as low accuracy. This approach does require extensive data or no previous data at all as the client will convey the requirements and does not depend on the historical data [19]. It is not a good practice as it may cause a delay in the development and delivery and may also force the development team to work overtime. The validation of this approach is based on the customer's budget and person-month factor.

On the contrary, the major advantage of the price-to-win approach is that it the estimation doesn't exceed the customer's budget. And as the functionality of the software is restricted with the customer's budget the quality of the software is compromised [19].

3.3. Learning Oriented Models

Machine Learning can be said to be a method of data analysis that is used to automate analytical model. It is also a branch of Artificial Intelligence (AI) which can be said to be based on the notion that machines such as robots and other computerized devices can learn from the data [20]. These approaches have the ability to learn from previous data and predict the future outcome based on the previous data. Some of the common machines learning algorithms that have been exploited in the software cost estimation are neural networks, fuzzy logic, genetic algorithms, Bayesian networks, support vector regression and analogy based. Researchers have proposed several machine learning software cost estimation models in order to improve the estimation accuracy [2]. And several studies have also reported that under the same model when the model is constructed with different historical project datasets or different experimental designs there is a variation in the predicted accuracy [2] [21] [22] [23] [24] [25].

A number of studies have concluded that machine learning models in software cost estimation outperform non-machine learning models [1] [2] [9] [10]. The use of these machine learning models in the software cost estimation process has gained significant popularity due to the wide margin of error in the classical estimation models [26]. There are remendous and continuous improvements in the machine learning algorithms which assists in achieving more accurate predictions when these are applied [2] [26]. These learning models consistently predict accurate results because of its learning nature from the previously completed projects. According to Monika *et al.* [10] investigation, it had been concluded that ANN was the prominent methods for developing estimating models. A brief overview of these models along with their strengths and weakness in the context of software cost estimation has been discussed. This will aid the researchers to understand which approach suits best for their methodology. Different machine learning models have different strengths and limitations and thus favor different estimation contexts [2].

3.3.1. Artificial Neural Networks

ANN is one of the major approaches that are exploited in the sector of machine learning models. As the name suggests, it is usually inspired by the neural part of the brain system with an intention of imitating an intelligent living organism [27]. It is composed of two layers, that is the input and output layers, within the layers there is a hidden layer which constitutes of units whose main purpose is to assign weights to the data that is fed from the input. These weights are assigned randomly to the data [28]. They are among the exploited tools in cost estimation due to their excellent performance [1]. Neural networks have existed for long as they can be traced back to the year 1940 although the best way to exploit them was the missing link which has been currently accomplished. There are different types of neural networks that exist with each of them having a defined level of complexity coupled with use. The most common and general type of neural network (NN) is the feed forward neural network as the name suggests data travels in only one direction which is from input to output [27]. Other types include Recurrent Neural Network (RN), Convolution Neural Networks (CNN), LTSM Recurrent Neural Networks, etc. The ANN approach is used in cost estimation where the pattern of certain data requires to be understood prior to estimating the project cost. In the cost estimating field, it is used to classify data into predefined classes, clustering, and prediction [28]. For instance, in projects such as the stock exchange market, it is used to make predictions in the market through the exploitation of the previous data.

Artificial Neural Networks are usually excellent in capturing non-linear relationships which makes it an iconic advantage of the model. The model has a deep neural net which requires fewer features [27]. The neural net has the ability to develop its own features which are also an advantage when limited features are available in the dataset for training the model. It is also flexible in the sense that there are a plethora of features one can choose from which include CNN's, RNN's, LSTM RNN's, etc. (**Figure 3**).

It has also some limitations whereby sometimes they tend to over-fit the data in the software cost estimation process [2]. Additionally, the model also requires an enormous amount of power for computation which may not be available at all times. It was better if the model is in a position to efficiently operate even in places where there is less consumption power as putting up the high computation power is costly to the involving firm.

Several Artificial Neural Networks models have been used in the software cost estimation processes which are mostly common according to the investigation [9]:

a) Feed-forward neural network

b) Recurrent Neural Network

c) Radial basis function (RBF) network

d) Neuro-fuzzy networks

Hamza *et al.* [9] concluded that choosing the right artificial neural network model is essential to get the accurate estimations. In their study [9] they have also concluded that feed-forward neural network works better than other models in ANN's but, the technique needs data filtering to prevent noise. In the case of noisy data, the radial basis function is more suitable.

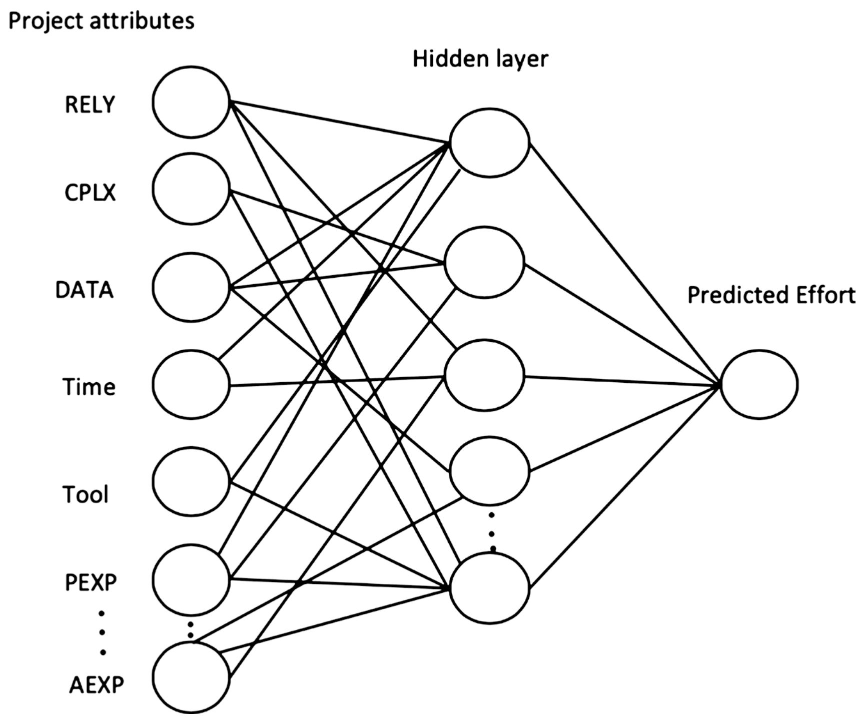

Figure 3. A neural network estimation model [3].

3.3.2. Genetic Algorithms

They are defined as adaptive and heuristic search algorithms which are a subject to the theory of natural selection by Darwin. In a current, study they are described as one of the most active areas of research which have been designed through nature-inspired metaheuristics [29]. Genetic Algorithm (GA) forms one of the soft computing techniques in software cost estimation process whereby its main role is to change certain parameters of classical methods such as COCOMO approach to predict software cost in a more accurate manner [29]. GA has widely been exploited in different fields of cost estimation such as correcting the identification system, path-searching problems within a project [29]. Additionally, GA has been used to solve a variety of NP-hard computational problems [29]. This model takes inspiration from nature such as bullet train design based on fish. Used for optimization problems (Np-problem) few of them are firefly algorithm particle swarm optimization and cuckoo search just to mention a few.

GA model usually exploits the optimization problem by using an evolutionary process. The first benefit of the model is the fact that it is easy to set up than neural networks but is less flexible. Once the algorithm begins it is on its own. It learns its own features; hence we don't have to supervise the process. On the other hand, it has a disadvantage which includes less flexibility, many hyper parameters which includes preference of functions, reproduction rates, the percentage of elitism and cross over, dealing with out of bound conditions, creating a strategy and setting the required tree sizes and depths within the model [30]. The model has no guarantee of finding an optimal solution, infinite time because it has asymptotic convergence, containing a number of parameters, sometimes the result is highly dependent on the parameters set. It has also self-adaptive parameters. It is computationally expensive and Meta models of functions are used in this process too.

The original use of the model was to establish manpower required to complete a certain project [29]. It is also used in scheduling different tasks within the field of cost estimation which implies that it has the ability to determine and evaluate a variety of tasks within a single project. It has also been used in mining data within the same scope which is considered as a complex exercise in case traditional methods are exploited in place. On the extreme end GA has been used in optimizing distributed tasks within the software cost estimation process. It has been used to solve distributed queries which imply that all relevant queries about a certain project can be accessed during the initial steps of its development. It is also easy to make assumption and predictions to the software being developed while using this method. It is somehow time conservative especially when operated by experienced personnel.

3.3.3. Fuzzy Logic

The model is a computing approach that is based on the degrees of truth rather than the unusual true or false normally referred to as Boolean logic which most of the modern computers are based on [31]. Additionally, it can be said to be an approach to computing based on many-valued function for instance, instead of a task completed or not, one can say 50% is completed. Fuzzy Logic (FL) approach gives an acceptable but definite output which is usually in response to inaccurate (fuzzy), distorted, incomplete and ambiguous input. FL was developed in 1965 by Loft Zadeh from a concept of fuzzy set theory. The first system is the one that is exploited in the estimation sector as most of the systems produce crisps data as input and expect the same type of data as output. There are three steps that have to be followed while using FL; the first is the Fuzzification which converts a crisp into a fuzzy set [31]. The second is the Fuzzy Rule-based System, at the step after all the crisp input has been fuzzified into their respective linguistic values the inference engine then derives their linguistic values [32]. Defuzzification is the final step which involves the conversion of fuzzy output into crisp output.

Fuzzy logic is deployed for decision making whereby it can be implemented with various sizes and abilities ranging from small microcontrollers to large workstation-station based software development. In the sector of software cost estimation, FL has been used to give acceptable reasoning although it does not guarantee accurate reasoning [32]. This approach is very easy to use despite the fact that it has numerous functions that it can execute when it comes to cost estimation of the software development [32]. Also, with the approach, it is advantageous as estimators of software are in a position to estimate and give an anticipation of the different aspects of the project even before the initiation level. The method has also a fast learning ability compared to other methods under the same scope [31]. The major disadvantage that the approach lacks guarantee of accuracy which is a vital part of the software cost estimation as to avoid incurring financial losses. Through the defuzzification process, the model is able to interact the output into a numerical value as per the desire.

There are a number of three applied approaches under this method which include; 2FA-kmodes it is used in clustering where numerical datasets are shown through fuzzy sets however the fuzz k-modes algorithm is used to maintain categorical attributes. The second approach is 2FA-kprototype it is used in clustering also where project data are clustered into homogeneous sets [31]. The final approach is the classical analogy it is used to predict the effort of the target project using information from former similar projects. The first two models 2FA-knodes, 2FA-kprototypes perform same and are both better than classical analogy. The similarity in the functionality of the two models is technically for the purposes of increasing efficiency within the model [32]. The major disadvantage associated with the two models is that they are not very good in COCOMO dataset and classical analogy doesn't recognize categories as "less risky", "medium", and "high". Thus, there is a need for implementations to be conducted for the model so as to enable future proper operation in case they are deployed in COCOMO and other datasets.

3.3.4. Bayesian Networks

The model is also referred to as a probabilistic directed acyclic graphical model.

It exploits graphical models to represent sets of variables coupled with their conditional dependencies through a directed acyclic graph (DAG) [33]. In other terms it uses Bayesian inference to perform probability computations they aim at modeling conditional dependence while developing software cost estimation which is usually represented by the edges within a directed graph. In addition to that, the model uses three main inference tasks: inferring unobserved variables, parameter learning and structure learning [33]. Through developers of software understanding the different relationships that exist in the model, they can efficiently conduct inference on random variables within software [33]. Each edge in the model corresponds to a conditional dependency while at the same time each node corresponds to a unique random variable. Thus, a specific pattern is usually followed in the model so as to attain its functionality within the software cost estimation process.

In a recent study, results have indicated that the inclusion of Bayesian Network, a machine-learning model in software cost estimation models it could improve on the levels of accuracy in the project [4]. This approach is able to break down project data about the cost into a form that is easy to analyze while at the same time it offers cost items coupled with the probability which can capture the uncertainty of different items within the software to be developed [33]. In software cost estimation sector, it is exploited in initial planning also as it has the ability to evaluate different parameters of planning such as time, cost coupled with resources to be exploited in the development of software. This model only allows exactly tested cost estimation information to be integrated with natural master feelings.

The best advantage with exploiting this model is that it gives a better assurance on accuracy and it is also able to improve on the overall quality of software that is being developed. Most of the software cost estimation models have some difficulties in offering assurance which makes the model outstanding compared to the rest. It is also a model that saves time which is of great importance when it comes to software project development [4]. Since the model encodes all variables, then it is also able to handle different types of missing data. In case it is deployed learning casual relationships, they help better understand a problem domain as well as forecast the consequences should there be an intervention [4]. Through probabilistic and casual semantics, it is ideal to use the model for representing prior data and knowledge.

Despite it being a complex approach in software cost estimation it is easy to use even with staffs who have less experience in the field [33]. On the other hand, there are some disadvantages while using this model such as Information theoretically infeasible it turns out that specifying a prior is extremely difficult challenge. Thus, being in a position to efficiently exploit this model, then users have to be well familiarized with the model's mode of learning. Being in a position to understand different languages, the model is able to execute more functions which are advantageous to the project running. Acquiring this knowledge takes some significant effort, this implies a lot of time will have to be consumed during the training period. Despite a lot of time being spent on training, also on the other hand it has to be the required training hence improper training would ruin the entire model. The models also do not always tell the truth as, they don't specify their actual prior, but they give credentials to the convenient one.

3.3.5. Support Vector Regression

It is also referred to as support vector networks or machines which are supervised models which are capable of learning algorithms and finally analyzing the same data that will be exploited for regression and classification analysis [34]. In other description, it is said to be a concept a set of related supervised learning methods usually used to analyze data coupled with recognized patterns. This model when applied in software cost estimation it takes a set of input data then it gives a prediction of each input [4]. The input must be a member of the support vector machine (SVM) which makes the model to be a non-probabilistic binary linear classifier [34]. In software cost estimation, it is used to solve problems related to pattern classification within the software. So as to apply it effectively in software cost estimation developers have to be in a position to design questions based on a problem and the design involved within the software.

The model has over the year's utilized regression and classification as the main mode of analyzing data within software. In China, the model was famously used in predicting software development efforts [34]. Support vector reasoning has been found to be helpful in hypertext and text categorization within the software. This model is usually utilized in software cost estimation so as to improve on the levels of accuracy compared to the classical query refinement schemes. It is also exploited in image segmentation in different software depending on the role that the software is designed to perform [34]. With the model in operation, the software development process automatically avoids over fitting of data. Thus, we can technically say that the model is conservative when it comes to financial and time while planning for a project.

A major advantage of the model is that it has the ability to work best with text data known as string kernel [4]. On the other hand, there it has been found to be occupying a huge memory and it does not operate well when the data set is huge. Occupying a huge memory concerning storage of data sets makes it inefficient a factor that they need to reconsider. The fact that it is occupying a huge data set for storage makes is not being able to meet the cost-effective criterion which should be a model for each model. In this model, there are implementations and improvements which have been done on it such as dimensionality reduction such as PCA and ICA. The implementation has been done on the model so as to improve on the functionality of the model.

3.3.6. Regression Tree

It is also referred to as a decision tree, is a logical model most exploited for decision analysis in software cost estimation. It is a common approach exploited in software cost estimation based on a number of factors and their consequences [35]. In other descriptions, the model is perceived as a procedure that is used for classification and regression in the sector of software cost estimation process [35]. The model is usually in the form of a tree structure with different internal nodes that stands for a test on an attribute. Individual branch on the tree usually represents an outcome of the test and the class label is held on the leaf node. Looking at the tree from a technical perspective, it is evident that all the branches stand for a specific function within the software development process. The model has usually trained through top-down training method which is specific to a certain direction [35]. However, current research shows there are efforts that are aimed at training the model so as to operate in a multidirectional purpose. This will be a breakthrough within the sector of software cost estimation as it means the model will be able to execute more functions than it does as for now [35]. Over the years it has been used to predict the amounts of effort required to develop a software system which has proved to be a good model as it saves on time and budgets that have been planned on. Thus, we can technically say that the model is cost efficient as it saves on the factors that are likely to jeopardize a developing project.

The regression tree has been widely used in inductive learning, in turn, it has proved to be good in terms of predictive accuracy especially in software cost estimation process despite its complexity [35]. One of the main advantages is the fact that it is fast compared to neural networks and support vector reasoning. The efficiency that is instilled in the model is because it has a simple structure which executes diverse roles is simple steps. Also, while exploiting the model they tend to be very interpretable thus one can almost make an anticipation of what is to be formed in the final project. This implies that there are possibilities of making assumptions on the outcome of the project that is being worked on.

3.3.7. Analogy Based

The analogy was used in the past as a method of supporting or supporting explanations of a particular natural phenomenon which was later incorporated in the software development sector of software development of for the purposes of software cost estimations [36]. Analogy Based is one of the most efficient approaches in software cost estimation due to its outstanding performance and capability of handling complex datasets [37]. The model exploits comparison as the main form of a subject to compare software project under considerations with past historical projects which have prior known characteristics, schedule and efforts. Thus, this model has to rely on previous projects similar to the ones that are to be developed thus the levels of accuracy are usually very high [37].

Conventional analogy-based models usually deploy the same number of analogies in all the projects that it is involved with similar data sets which improves on the estimation approximations. There are some researchers that claim using the same number of analogies could only improve on the data sets only and not the entire project. Thus, when exploiting this model, it is important to understand the

Table 1. Software cost estimation techniques comparison.

N0	Method	Type	Strengths	Weakness	Accuracy	Data	Validation
1	COCOMO-II	AM	Simple to carryout estimations; takes less time and effort to estimate; Useful in large projects	Details of the past projects needed to estimate; May leave out hidden costs hence lead to more expensive estimation; Calibration is required; Cannot meet current software standards	MA	ED	EV
2	FPA	AM	Easy to estimate development costs at the requirements gathering stage/initial stage; Independent of languages and tools used	Quality attributes, development time and man power are not considered	MA	LD	EV
3	PM	AM	Estimation depends on only time and size which are key to cost estimation; Needs fewer parameters compared to COCOMO 81 and COCOMO II	Does not consider other important aspects of the software development	MA	LD	EV
4	EJ	NA	Best technique where limited data is available; Experience of experts makes the estimation more accurate and realistic	Biased opinions of the experts; Difficult to document the parameters used by the experts to estimate; Experts require experience of similar projects	LA	LD	EV
5	TDE	NA	Only few details are required to estimate; Simple and less time consuming	Difficult to identify low level problems which causes under estimation; Less details may overlook important attributes of the project	LA	LD	n/a
6	BUE	NA	More stable compared to top down approach; Errors are estimated and very stable	Development time and system-level activities are not considered	LA	LD	n/a
7	PTWE	NA	It depends only on the customer budget	Costs do not accurately reflect the work required; Low quality system is developed due to client budget constraint	LA	ND	n/a
8	NN	LM	Very good in capturing non-linear relationships; Deep neural Nets do not require a lot of features, they come up on their own; There is a lot of flexibility, you could choose different architectures; RNN's, LSTM, etc.	They tend to over fit; They require enormous amount of computation power	VHA	ED	NI, DCS, CVM
9	GA	LM	Need not be supervised	Too many hyper parameters	HA	ED	CVM, VAF, RWS
10	FL	LM	Considers real valued states instead of binary	Does not perform well on complex datasets like COCOMO	VHA	ED	JM, DCS, CVM
11	BN	LM	Bayesian network encodes all variables; missing data entries can be handled	Difficult to model	HA	ED	CVM
12	SVR	LM	It works best with text data: string kernel	It takes a lot of memory (RAM); Doesn't scale well when dataset is huge	HA	ED	LOM
13	RT	LM	more capable of handling noisy datasets	Models are unstable at times, suffer with high variance, low bias. (Keyword: bias-variance tradeoff)	MA	LD	CVM, DCS, LOM
14	ABE	NA	Simple and easy to use; No bootstrap cost	They do not work with categorical data	MA	LD	EV, ED

characteristics of each data sets so as to be able to discover the optimum set of analogies for each project [38]. Accuracy levels can only be increased when users of the model are aware of the different data characteristics instilled by each data set. Through the comparison of various datasets, the model is used to approximate the time that a project would take for it to be completed. In cost estimation, the model is relied on to give empirical evidence on the probability of certain software features being accurate.

It is also flexible and intuitive in nature as it can be applied in a variety of circumstances where other algorithmic modeling and estimating techniques don't operate [36]. As much as it is said to be dependent upon past history so as to execute its functions, it can still operate even where past data is not available. However, there are still some setbacks while using the model as it lacks appropriate analogs [38]. For instance, despite it being widely used in the industries it has not stipulated standards on how it should be exploited for expert opinion-based opinion. The common advantage that most users of the method are familiar with is the ability of the model to avoid bootstrapping of cost [38]. On the extreme end, it is accompanied by a major disadvantage which is its inability to operate with categorical data. Hence more implementations need to be done on the model so as to enable to operate with all types of data (**Table 1**).

4. Conclusion & Future Work

In this paper, we presented an overview of the software cost, effort and size estimation techniques based on algorithmic, non-algorithmic and learning-oriented approaches. We also tabulated all the techniques based on their type, strengths, weaknesses, amount of data and validation methods used by them. Our major findings from the previous studies show that the Neural Network based models outperform other cost estimation techniques in terms of accuracy followed by fuzzy logic [39]. Our survey also shows that no one technique is perfect and all of them have their own advantages and disadvantages. We recommend the researchers and practitioners to choose the best technique which fits their best needs. This study may also give an insight into the software cost estimation techniques for the researchers who are new to this area. We found that there has not been much research on estimating the development time or schedule for a project. Our future work may involve researching the application of the LSTM Recurrent Neural Networks model in predicting the time series for the project.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Pinkashia, S. and Singh, J. (2017) Systematic Literature Review on Software Effort Estimation Using Machine Learning Approaches. 2017 *International Conference on Next Generation Computing and Information Systems*, Jammu, India, 11-12 De-

cember 2017, 43-47. https://doi.org/10.1109/ICNGCIS.2017.33

[2] Wen, J., Li, S., Lin, Z., Hu, Y. and Huang. C. (2012) Systematic Literature Review of Machine Learning Based Software Development Effort Estimation Models. *Information and Software Technology*, **54**, 41-59. https://doi.org/10.1016/j.infsof.2011.09.002

[3] Barry, B., Abts, C. and Chulani, S. (2000) Software Development Cost Estimation Approaches—A Survey. *Annals of Software Engineering*, **10**, 177-205. https://doi.org/10.1023/A:1018991717352

[4] Moloekken-ØEstvold, K., Jørgensen, M., Tanilkan, S.S. Gallis, H., Lien, A.C. and Hove, S.W. (2004) A Survey on Software Estimation in the Norwegian Industry. 10*th International Symposium on Software Metrics*, Chicago, IL, 11-17 September 2004, 208-219.

[5] Li, M.-S., He, M., Yang, D., Shu, F.-D. and Wang, Q. (2007) Software Cost Estimation Method and Application. *Journal of Software*, **18**, 775-795.

[6] Magne, J. and Shepperd, M. (2007) A Systematic Review of Software Development Cost Estimation Studies. *IEEE Transactions on Software Engineering*, **33**, 33-53. https://doi.org/10.1109/TSE.2007.256943

[7] Tomás, V., Ochoa, S.F. and Perovich, D. (2017) Survey of Software Development Effort Estimation Taxonomies. Technical Report. Pending ID. Computer Science Department, University of Chile, Chile.

[8] Rajeswari, K. (2018) A Critique on Software Cost Estimation. *International Journal of Pure and Applied Mathematics*, **118**, 3851-3862.

[9] Haitham, H., Kamel, A. and Shams, K. (2013) Software Effort Estimation Using Artificial Neural Networks: A Survey of the Current Practices. 2013 10*th Information Technology*: *New Generations*, Las Vegas, NV, 15-17 April 2013, 731-733. https://doi.org/10.1109/ITNG.2013.111

[10] Sangwan, O.P. (2017) Software Effort Estimation Using Machine Learning Techniques. 2017 7*th International Conference on Cloud Computing*, *Data Science & Engineering-Confluence*, Noida, India, 12-13 January 2017, 92-98.

[11] Boehm, B.W. (1981) Software Engineering Economics. Prentice-Hall, Englewood Cliffs, NJ.

[12] Barry, B., Clark, B., Horowitz, E., Westland, C., Madachy, R. and Selby, R. (1995) Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. *Annals of Software Engineering*, **1**, 57-94. https://doi.org/10.1007/BF02249046

[13] IFPUG, FPCPM (2000) International Function Point Users Group (IFPUG) Function Point Counting Practices Manual.

[14] Komal, G., 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**, 265-572.

[15] Putnam, L.H. (1978) A General Empirical Solution to the Macro Software Sizing and Estimating Problem. *IEEE Transactions on Software Engineering*, **4**, 345-361. https://doi.org/10.1109/TSE.1978.231521

[16] Warburton, R.D.H. (1983) Managing and Predicting the Costs of Real-Time Software. *IEEE Transactions on Software Engineering*, **5**, 562-569. https://doi.org/10.1109/TSE.1983.235115

[17] Kemerer, C.F. (1987) An Empirical Validation of Software Cost Estimation Models. *Communications of the ACM*, **30**, 416-429.

https://doi.org/10.1145/22899.22906

[18] Christopher, R. and Roy, R. (2001) Expert Judgment in Cost Estimating: Modelling the Reasoning Process. *Concurrent Engineering*, **9**, 271-284. https://doi.org/10.1177/1063293X0100900404

[19] Hareton, L. and Zhang, F. (2002) Software Cost Estimation. In: *Handbook of Software Engineering and Knowledge Engineering*: *Volume* 2: *Emerging Technologies*, World Scientific Publishing Co Pte Ltd., Singapore, 307-324. https://doi.org/10.1142/9789812389701_0014

[20] Sharma, S. (2017) Applications of Genetic Algorithm in Software Engineering, Distributed Computing and Machine Learning. *International Journal of Computer Applications & Information Technology*, **9**, 208-212.

[21] Gray, A.R. and Macdonell, S.G. (1999) Software Metrics Data Analysis—Exploring the Relative Performance of Some Commonly Used Modeling Techniques. *Empirical Software Engineering*, **4**, 297-316. https://doi.org/10.1023/A:1009849100780

[22] Ross, J., Ruhe, M and Wieczorek, I. (2000) A Comparative Study of Two Software Development Cost Modeling Techniques Using Multi-Organizational and Company-Specific Data. *Information and Software Technology*, **42**, 1009-1016. https://doi.org/10.1016/S0950-5849(00)00153-1

[23] Abbas, H. (2002) Comparison of Artificial Neural Network and Regression Models for Estimating Software Development Effort. *Information and Software Technology*, **44**, 911-922.

[24] Dolado, J.J. (2001) On the Problem of the Software Cost Function. *Information and Software Technology*, **43**, 61-72. https://doi.org/10.1016/S0950-5849(00)00137-3

[25] Ingunn, M. and Stensrud, E. (1999) A Controlled Experiment to Assess the Benefits of Estimating with Analogy and Regression Models. *IEEE Transactions on Software Engineering*, **25**, 510-525. https://doi.org/10.1109/32.799947

[26] Ahmed, B.M. (2018) Predicting Software Effort Estimation Using Machine Learning Techniques. 2018 8*th International Conference on Computer Science and Information Technology*, Amman, 11-12 July 2018, 249-256. https://doi.org/10.1109/CSIT.2018.8486222

[27] Poonam, R. and Jain, S. (2016) Enhanced Software Effort Estimation Using Multi Layered Feed Forward Artificial Neural Network Technique. *Procedia Computer Science*, **89**, 307-312. https://doi.org/10.1016/j.procs.2016.06.073

[28] Idri, A., Khoshgoftaar, T.M. and Abran, A. (2002) Can Neural Networks Be Easily Interpreted in Software Cost Estimation? 2002 *IEEE World Congress on Computational Intelligence*. 2002 *IEEE International Conference on Fuzzy Systems*, Honolulu, HI, 12-17 May 2002, 1162-1167.

[29] Singh, B.K. and Misra, A.K. (2012) Software Effort Estimation by Genetic Algorithm Tuned Parameters of Modified Constructive Cost Model for NASA Software Projects. *International Journal of Computer Applications*, **59**, 22-26.

[30] Burgess, C.J. and Lefley, M. (2001) Can Genetic Programming Improve Software Effort Estimation? A Comparative Evaluation. *Information and Software Technology*, **43**, 863-873. https://doi.org/10.1016/S0950-5849(01)00192-6

[31] Anupama, K., Soni, A.K. and Soni, R. (2013) Radial Basis Function Network Using Intuitionistic Fuzzy C Means for Software Cost Estimation. *International Journal of Computer Applications in Technology*, **47**, 86-95.

https://doi.org/10.1504/IJCAT.2013.054305

[32] Anish, M., Parkash, K. and Mittal, H. (2010) Software Cost Estimation Using Fuzzy Logic. *ACM SIGSOFT Software Engineering Notes*, **35**, 1-7. https://doi.org/10.1145/1668862.1668866

[33] Hrvoje, K. and Gotovac, S. (2015) Estimating Software Development Effort Using Bayesian Networks. 2015 23*rd International Conference on Software*, *Telecommunications and Computer Networks*, Split, Croatia, 16-18 September 2015, 229-233. https://doi.org/10.1109/SOFTCOM.2015.7314091

[34] Bhavendra Kumar, S., Sinhal, A. and Verma, B. (2013) A Software Measurement Using Artificial Neural Network and Support Vector Machine. *International Journal of Software Engineering & Applications*, **4**, 41-52. https://doi.org/10.5121/ijsea.2013.4404

[35] Magne, J. (2004) Regression Models of Software Development Effort Estimation Accuracy and Bias. *Empirical Software Engineering*, **9**, 297-314. https://doi.org/10.1023/B:EMSE.0000039881.57613.cb

[36] Ali, I., Amazal, F.A. and Abran, A. (2016) Accuracy Comparison of Analogy-Based Software Development Effort Estimation Techniques. *International Journal of Intelligent Systems*, **31**, 128-152. https://doi.org/10.1002/int.21748

[37] Hathaichanok, S. and Prompoon, N. (2012) Framework for Developing a Software Cost Estimation Model for Software Modification Based on a Relational Matrix of Project Profile and Software Cost Using an Analogy Estimation Method. *International Journal of Computer and Communication Engineering*, **1**, 129-134. https://doi.org/10.7763/IJCCE.2012.V1.36

[38] Manikavelan, D. and Ponnusamy, R. (2015) Improvised Analogy Based Software Cost Estimation with Ant Colony Optimization. *Research Journal of Applied Sciences*, *Engineering and Technology*, **10**, 293-297. https://doi.org/10.19026/rjaset.10.2490

[39] Papatheocharous, E. and Andreou, A.S. (2012) Software Cost Modelling and Estimation Using Artificial Neural Networks Enhanced by Input Sensitivity Analysis. *Journal of Universal Computer Science*, **18**, 2041-2070.

Abbreviations

AM	Algorithmic Method
NA	Non-Algorithmic Method
LM	Learning-Oriented Method
LA	Low Accuracy
MA	Moderate Accuracy
HA	High Accuracy
VHA	Very High Accuracy
ND	No Data
LD	Limited Data
ED	Extensive Data
CVM	Cross Validation Method
NI	Number of Iterations
VAF	Variance Accounted For
RWS	Roulette Wheel Selection
JM	Jackknife Method
LOM	Leave-one-out Method
DCS	Degree of Confidence and Significance
EV	Empirical Validation
ED	Euclidean Distance
WBS	Work Breakdown Structure
n/a	Not Applicable
COCOMO	Cost Constructive Model
FPA	Function Point Analysis
PM	Putnams's Model
EJ	Expert Judgement
TDE	Top Down Estimation
BUE	Bottom Up Estimation
PTWE	Price-To-Win Estimation
NN	Neural Networks
GA	Genetic Algorithm
FL	Fuzzy Logic
BN	Bayesian Networks
SVR	Support Vector Regression
RT	Regression Tree
ABE	Analogy Based Estimation