International Journal of Finance and Managerial Accounting, Vol.5, No.18, Summer 2020

A Model-Driven Decision Support System for Software Cost Estimation (Case Study: Projects in NASA60 Dataset)

Hassan Rashidi
Department of Statistics, Mathematics and Computer Sciences, Allameh Tabataba'i University
hrashi@atu.ac.ir

ABSTRACT

Estimating the costs of software development is one of the most important activities in software project management. Inaccuracies in such estimates may cause irreparable loss. A low estimate of the cost of projects will result in failure on delivery on time and indicates the inefficiency of the software development team. On the other hand, high estimates of resources and costs for a project will waste opportunities for other projects. This paper presents a methodology for estimating the costs of software development. The methodology is a model-driven decision support system that consists of four subsystems; namely Data subsystem, Model subsystem, User Interface subsystem, and Knowledge subsystem. The core supports of this system are based on coherent theory on the nature of collaborative work and their mathematical models in software engineering that included in the model subsystem. This core provides a theoretical foundation for decision optimizations on the optimal labor allocation, the shortest duration determination, and the lowest workload effort and costs estimation. The experimental results and evaluations on Dataset NASA60 show that the proposed system has significant conformance with experience in practice. Based on the proposed decision support system, a wide range of fundamental problems in software project organization and cost estimation can be solved rigorously.

Keywords:

Software Development Management, Decision Support System, Optimization

1. Introduction

One of the major challenges for software companies, nowadays, is to estimate the cost and labor requirements for their projects. Accurate cost-estimation of software projects optimizes the internal and external processes, staff works, efforts, and the overheads to be coordinated with one another. In the management software projects, estimation must be taken into account so that it reduces costs, timing and possible risks to avoid project failure. The business environment and climate for software companies are constantly changing, and it is becoming more and more complex. Software developing organizations in both private and public sectors are under pressures that force them to respond quickly to changing conditions and to be innovative in the way they operate. Such activities require these organizations to be agile and to make frequent and quick strategic, tactical, and operational decisions, some of which are very complex. Making such decisions may require considerable amounts of relevant data, information, and knowledge. Processing these, in the framework of the needed decisions, must be done quickly, frequently in real-time, and usually requires some computerized supports.

The main motivation, here, is to provide a foundation for estimating software costs in terms of labor and time required for the development. This foundation is created in the form of a decision support system. This system provides an essential tool for software managers. Its essentially is due to the cost of the software developed is already known, whereas the costs of development of new software is always a challenging problem and matter of judgment. The initial estimation is an important factor for project management because it reduces the margin of error in estimating software cost. This is particularly true when relatively large software components such as subsystems are reused.

The cost of a software project is usually perceived as a function of the symbolic size of the project in the term of lines of code in the source. The labor and time allocations for a given workload of a project cannot be carried out freely, rather than be constrained by certain laws as defined in the decision support system of software engineering costs. The main focus of the system is to provide a theoretical foundation for software engineering decision optimizations in the following processes to: (a) estimate project size; (b) determine the ideal workload, (c) allocate optimal labor, (d) determine the shortest duration, (e) determine the project cost, and (f) derive the project cost. The proposed system is abbreviated by DSSSCE (Decision Support System for Software Cost Estimating).

The structure of the remaining sections is as follows: Section 2 makes a literature review over the matter. Section 3 presents the methodology used in this research with details. Section 4 presents the results of implementing the methodology. Finally, Section 5 is considered for the discussions and conclusion.

2. Literature Review

Software development in both cost and labor estimation is an indispensable component of the software development process so that an accurate assessment of development cost is an important guarantee for the success project. However, there are many insufficient in current assessment methods. Recently, the decision support system is implemented and used for several aspects of software management. In this section, we review the main contribution of incorporating decision support system into software project management.

Donzelli applied Decision Support System (DSS) to software engineering8). The research proposes exploiting the advantages of the three traditional modeling methods (analytical models and continuous and discrete-event simulation) by combining them into a hybrid two-level modeling approach. As a case study, the research used this hybrid approach to model the NASA SEL software process. The model focuses on the main process quality attributes (effort, delivery time, productivity, rework percentage, and product defect density) and numerous sub-attributes (final product size, process staffing profile, staffing profiles over single activities, defect patterns, and so on). Because no single modeling approach is well suited to representing all process aspects, hybrid simulation is emerging as a promising approach.

Li analyzed software development cost estimation methods13). This research brands full use of COCOMO2, Function Point Analysis method, and COCOMO2 with the Fuzzy Delphi method to put developments and applications of a decision support system for software project cost estimation forward. The experimental application implies that the system is viable, the result of development cost estimation is

accurate, and can effectively support the analysis of development cost.

Pashaei Barbin and Rashidi proposed a decision support system using a combination of multi-layer artificial neural network and decision tree to estimate the cost of software projects (Pashaei Barbin, and Rashidi, 2015). In the model included in the proposed system, the normalizing factors are vital in evaluating efforts and costs estimation, which is carried out using the C4.5 decision tree. Moreover, the testing and training factors are done by a multi-layer artificial neural network, and the most optimal values are allocated to them. The experimental results and evaluations on Dataset NASA60 show that the proposed system has less amount of the total average relative error compared with that of COCOMO model.

Monika and Sangwan used Machine learning techniques to accurately predict software effort values 20). This paper presents a review of various machine-learning techniques using in the estimation of software project effort, namely Artificial Neural Network, Fuzzy logic, Analogy estimation, etc. Machine learning techniques are consistently predicting accurate results because of its learning natures from previously completed projects. Moreover, this research summarizes that each technique has its own features and behaves differently according to the environment so no technique can be preferred over each other.

Khan et al. (2018) used a new meta-heuristic algorithm inspired by the strawberry plant for optimization of COCOMO effort estimation method 33)(Khan et al., 2018). In this research, the NASA 93 data set is used in the experiments. The Magnitude of Relative Error (MRE) and Mean Magnitude of Relative Error (MMRE) is evaluated. The experimental results of the proposed algorithm with the COCOMO model shows a decline in MMRE to 23.8%.

Venkataiah et al. (2019) proposed a hybrid methodology for tuning parameters of the COCOMO model (Venkataiah, Mohanty, and Nagaratna, 2019), consisting of two phases. The COCOMO 81, IBMDPS, COCOMO NASA 2 and DESHARNAIS are used to test the performance of the proposed model. The K-means clustering procedure is used to make different clusters is the first phase of the hybrid approach. In the second phase, Particle Search Optimization (PSO) used for tunning values of COCOMO on different clustered data. The experimental results found that MARE and RMSE results are outperforming compared to others.

Bajta and Idra (2019) used a design method to organize the knowledge identified as a cost estimation taxonomy for Global Software Development (El Bajta, and Idri, 2019). The proposed taxonomy offers a classification scheme for the cost estimation of distributed projects. The cost estimation taxonomy consists of four dimensions: cost estimation context, estimation technique, cost estimate, and cost estimators. Each dimension, in turn, has multiple facets. The taxonomy could then be used as a tool for developing a repository for cost estimation knowledge.

In 29), an improved approach was proposed to software cost estimation using an output layer self-connection recurrent neural networks (OLSRNN) with kernel fuzzy c-means clustering (KFCM). The proposed OLSRNN method follows the basics of traditional RNN models for integrating self-connections to the output layer; thereby, the output temporal dependencies are better captured. In this research, five publicly available software cost estimation datasets are adapted to verify the efficacy of the proposed KFCM-OLSRNN method using the validation metrics such as MdMRE, PRED (0.25), and MMRE. The experimental results proved the efficiency of the proposed method.

Rivadeh and Khadivar (2019) used the system dynamics approach and fuzzy logic for software cost modeling (Rivadeh, and Khadivar, 2019). This research consists of two statistical populations. The first statistical population includes IT managers and software development project managers. The second population includes experts from Magfa Company. The tool used to collect data is a questionnaire. The data and information related to the project of development of business intelligence software were at Magfa Company and have been gathered from the people involved in the project. After simulating and testing the model, three scenarios have been defined to reduce software development costs, which include: increasing personnel experience and increasing the experience of project managers, increasing the capabilities and competencies of manpower, and changing the lifecycle of the system from cascading to agile methodology scenario. The findings indicate that the company will see a further reduction in software development costs by using the agile life cycle model.

Parthasarathi and Rajnish (2019) studied an empirical high-performance interpolation model to estimate the effort of the software projects31)(Parthasarathi, and Rajnish, 2019). This model compares with the COCOMO based equations and predicts its result analyzing individually taking different cost factors. The equation consists of one independent variable (KLOC) and two constants a, b which are chosen empirically taking different NASA projects historical data and the results viewed in this model are compared with COCOMO model with different scale factor values.

Naik and Nayak (2019) introduced an intellectual model of a software cost model that is mainly targeted to perform optimization of entire cost estimation modeling by incorporating a predictive approach (Naik, and Nayak, 2019). Powered by the deep learning approach, the outcome of the proposed model is found to be cost-effective in comparison to existing cost estimation modeling.

Prasad et al. (2019) introduced a new software EDC estimation process, namely Regression testing based software EDC estimation technique (Prasad, Sreenivas, and Veena, 2019). The experimental analysis is carried out on two datasets namely NASA 93 and COCOMO datasets. The proposed regressing testing model would generate various test cases by comparing the attribute values of these datasets to predict the quality of the software in terms of effort, duration, and cost. Based on these test case values, software project estimation can be done efficiently. In this work, an adaptive firefly algorithm is utilized for the efficient test case generation which would combine the multiple attributes of the dataset to generation optimal test cases with the concern of ranking. The overall evaluation of the research is conducted on the java simulation environment and proved that the proposed research technique leads to ensure the optimal outcome than the existing research techniques.

3. Methodology

The methodology used in this research is based on decision support systems. These systems are a class of information systems (including but not limited to computerized systems) that support business and organizational decision-making activities. A properly designed DSS is an interactive computer-based system intended to aid decision-makers to compile valuable information from a combination of raw data, personal knowledge, documents, or business models to identify and solve problems. The supports given by DSS can be separated into three distinct, interrelated categories: Personal Support, Group Support, and Organizational Support. The DSS components may be as: (a) Inputs: Factors, numbers, and characteristics to analyze; (b) User Knowledge and Expertise: Inputs requiring manual analysis by the user; (c) Outputs: Transformed data from which DSS "decisions" are generated; and (d) Decisions: Results generated by the DSS based on user criteria.

There are several ways to classify DSS applications. Not every DSS fits neatly into one category, but a mix of two or more architectures in one. The taxonomy for DSS was created by Daniel Power 7). Using the mode of assistance as the criterion, power differentiates communication-driven DSS, data-driven DSS, document-driven DSS, knowledge-driven DSS, and model-driven DSS. The AIS SIGDSS 1) classifies DSS into: (a) Communications-driven and group DSS (GSS); (b) Data-driven DSS; (c) Document-driven DSS; (d) Knowledge-driven DSS, Data Mining, and Management Expert Systems Applications; and (e) Model-Driven DSS; and (f) Compound DSS. A compound DSS is the most popular classification for a DSS. It is a hybrid system that includes two or more of the five basic structures described by Holsapple and Whinston.

Figure 1 shows the general components and structure of DSS 21)(Sharda, Aronson, and Turban, 2015). As we can see in the figure, there are four basic subsystems: (a) the data management subsystem; (b) the model management subsystem; (c) the user-interface (dialog) subsystem and (d) the knowledge-based management subsystem. These subsystems are described in the following subsections.

The 'User interface' refers to the way a manager or decision-maker can use the system to support his/her decision-making needs without having to become an expert in its technology. Figure 2 shows the components of the User Interface Subsystem in the DSS. In fact, inside the DSS, the flow of information from the user to the system and from the system to the user is handled by the User Interface Management Subsystem (UIMS). The UIMS processes user commands, issued in whatever action language, and passes the responses to the data and model

management subsystems. On the contrary direction, it presents information from those subsystems to the user. Increasingly, the action language could be based on Web or operating system GUI concepts. It may also include some natural language processing capabilities. The main inputs and outputs of the decision support system, used for software cost estimation, are summarized in Table 1.

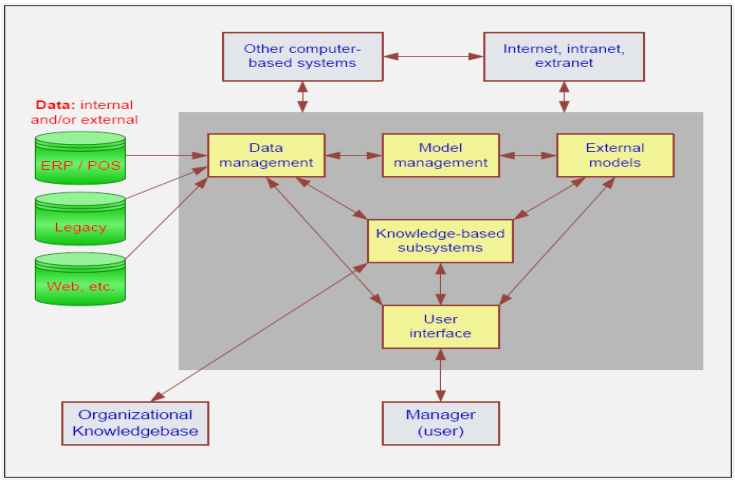

Figure 1- The components and structure of each DSS component, in general
(Sharda et al. Aronson, and Turban, 2015)

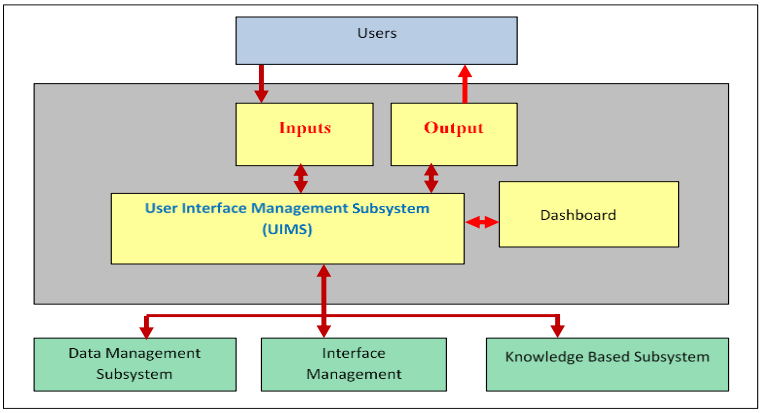

Figure 2- The components of User Interface Subsystem

Table 1- The main inputs and Output Variables of the Decision Support System

Inputs Variables	Output Results
The Number of Persons Required in The Project-LR	Estimate Project Size by the expert - $\overline{S_{TD}}$
The Time Spent in the Project-TS	Estimate Project Size from the subsystems and components- $\overline{S_{BU}}$
The Maximum Size of the Project –Smax	The Ideal Workload- IWL
The Minimum Size of the Project –Smin	Optimal Labor Allocation- LR_0
The Expected Size of the Project –Sexp	Shortest Duration- TS_{min}
The Number of Subsystems in the Project –n	Minimize Project Effort- WL_{Min}
The Number of Components in each Subsystem –m	Optimized Project Cost- CP_{min}

The 'User Interface' (UI) connects the user to the other subsystems. The knowledge-based subsystem, in addition to being connected to the user via the UI, may also connect to the DBMS to obtain the data it needs, to external models, and an organizational knowledge base. The UIMS processes user commands, issued in any action language that it requires, and passes them on to the data and model management subsystems. In the reverse direction, it presents information from those subsystems to the user.

The 'Data' refers to the information needed to make a decision, generally stored in a database, and to how these data are organized and managed by a DBMS. Figure 3 shows the components of the Data Management Subsystem in the DSS. These components are 'DSS database', 'DBMS', 'Data directory' and 'Query facility'.

The primary functions and capabilities of DBMS are storage, retrieval, and control. The DBMS must manage the database to organize, extract/access, modify, delete, and catalog data. The role of 'Extraction' is the process of capturing data from multiple sources, filtering them, condensing, summarizing, and reorganizing the data to load into a DSS database such as a data warehouse. The function of the query facility, in building and using DSS, is often to access, manipulate, and query data.

The 'Query facility' performs several tasks. It accepts the requests for data from other DSS components, determines how the requests can be filled (consulting the data directory if necessary), formulates the detailed requests, and returns the results to the application. The function of a 'Data directory' is to provide a catalog of all data in the database. It includes the data definitions and other information needed to facilitate and control access to data via the DMBS.

The Internal Data Sources are categories into five groups, related to the size of components in the subsystems used in the organization, real data collected from the completed projects as well as the rate of coordination between personals employed in the software development teams.

The knowledge subsystem can either supply the required expertise for solving some aspects of the problem or provide some knowledge to enhance the operation of other DSS components. It can help in model selection by capturing the knowledge of experts as to the applicability of different models in various situations, making it available to people having less expertise in this area. It is often used to better manage other DSS components. The components in this subsystem may be expert knowledge, neural networks, intelligent agents, fuzzy logic case-based reasoning, and so on.

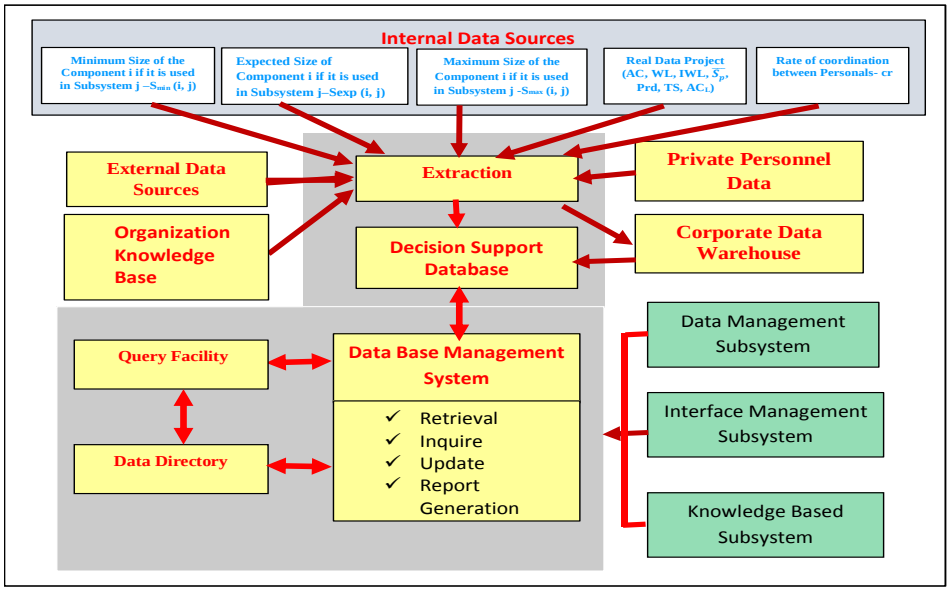

Figure 3- The components of the Data Subsystem (the parts below of the figure is taken from 21))

The *Models* refer to the models used to analyze the data and to forecast the results of a decision, as well as to the software used to manage the models in a DSS. Figure 4 shows the components of the model management Subsystem in the DSS. These components are Model Base, MBMS, Modeling language, Model directory, Model Execution, Integration, and Command Processor.

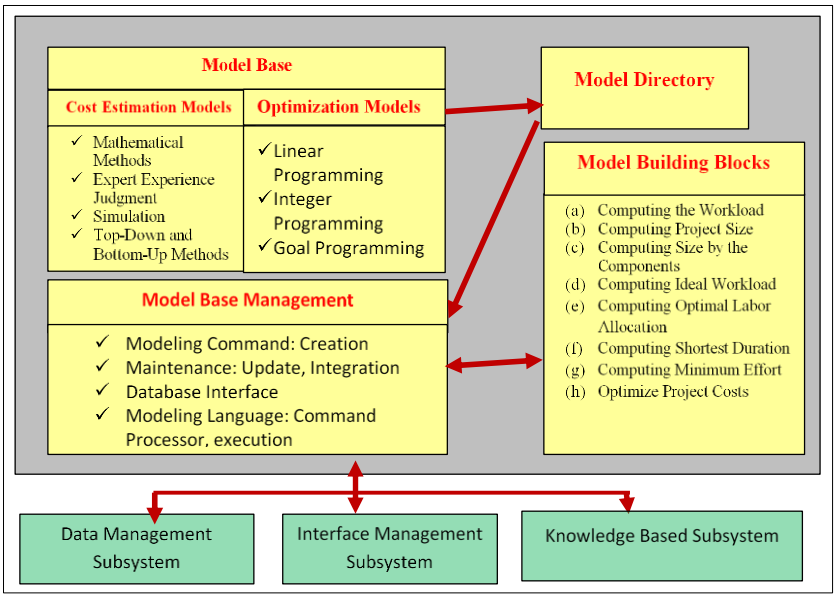

Figure 4- The components of the Model Subsystem(the part 'Model Base Management' is taken from (Sharda, Aronson, and Turban, 2015))

The 'Model base' contains cost estimation models and some optimization models. These models relate to cost budget preparation, manpower demand, personnel employment, and process arrangement, which are an important preparatory work to improve the efficiency of software development and support the software project implementation by low price reasonably.

There are many software scale estimation methods](Qi, and Boehm, 2017), (Pressman, 2014), (Sommerville, 2018)], the commonly used method is based on mathematical method, expert experience judgment, simulation, top-down and bottom-up methods. These models based on mathematical methods known as parameters or statistical models, which are the most interesting methods in software scale estimation. In the optimization models, some companies are intended to identify the best decision, given specific constraints. The majority of the cost estimation in practical models in use is obtained by the regression techniques.

The 'Model base Management' has five functions: (a) Model creation, using some programming languages, DSS tools and/or subroutines, and other building blocks; (b) Generation of new routines and reports; (c) Model updating and changing; (d) Model data manipulation; (e) The Model execution, integration, and command. The 'Model directory' is similar to a database directory. It is a catalog of all the models and other software in the model base. It contains model definitions, and its main function is to answer questions about the availability and capability of the model.

The model building block is a repository that can be used especially for the estimation of cost and labor required for the project. More specifically, they are equations that used for (a) estimate project size; (b) determine the ideal workload, (c) allocate optimal labor, (d) determine the shortest duration, (e) determine the project cost, and (f) derive the project cost.

The proposed methodology used in this research is a type of Model-Driven DSS in which the model building blocks models are core. We systematically present the equations and calculations of the model building block in the subsystem model (See Fig. 3).

- **(a) Computing The workload**: If we make a top-down approach to compute the workload in a project, we must determine the number of personals required (LR) and the actual time spent on the project (TS). These values are specified by an expert with some consideration of overhead of interpersonal coordination in costs. The workload of the project is the product of LR×TS. If we want to make the relationship between the workload WL of the project with the ideal workload of the project, Eq. (1) can be used [(Pressman, 2014), (Rashidi, 2014), (Wang, 2006a)]:

$$\begin{aligned} WL = LR \times TS &= IWL(1 + ov) \\ &= IWL_1 \times cr \times \frac{LR \times (lR - 1)}{2} \end{aligned} \quad (1)$$

[Person-Month]

Where IWL is the ideal workload without the overhead or that of a single person project, *ov* is the overhead coefficient and *rc* is the rate of coordination between personals in the project.

- **(b) Computing Project Size:** In the top-down approach, knowing the size of the software is the starting point of cost estimation. The project size is usually represented by the symbolic size S_{TD} of software in the unit of a thousand lines of code (kLOC). The Project size S_{TD} can be estimated by a weighted average of its symbolic size, $\overline{S_{TD}}$, as follows [(Pressman, 2014), (Rashidi, 2014)]:

$$\overline{S_{TD}} = \frac{(Smax + 4Sexp + Smin)}{6} \quad (2)$$

[kLOC]

where S_{exp} is the most likely expectation of the size of the project, S_{max} and S_{min} are the maximum or minimum expectation, respectively, that are obtained as inputs from the expert. In Eq. 2, it can be seen that the weighted average size estimation gives a higher weight to the most likely expectation. The size of the project determined by Eq. (2) is a reasonably accurate technique when empirical data are available on similar projects as references.

- **(c) Computing Size by the components**: Since the empirical comparability is not always available at the whole project level in software engineering, a more generic approach to size estimation is to use a bottom-up approach and the strategy of division and conquer. Usually, we assume a software project encompasses n subsystems, and each subsystem consists of m components. Therefore, the size of the project can be estimated as a sum of the weighted average of estimated sizes of all components, $\overline{S_{BU}}$, according to Eq. (3) [(Pressman, 2014), (Rashidi, 2014)].

$$\overline{S_{BU}} = \sum_{i=1}^{n} \sum_{j=1}^{m} \overline{Sij} = \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{(Smax(ij) + 4Sexp(ij) + Smin(ij))}{6} \quad (3)$$

[kLOC]

- **(d) Computing Ideal Workload:** Once the size of a given project is obtained by Eq. (3), the workload WL can be determined on the basis of software productivity benchmarks of the industry or a specific organization. The workload of the software project is determined by the Eq. (4) as the ratio of the estimated project size $\overline{S_{PrJ}}$ (i.e, the compliance between $\overline{S_{BU}}$ and $\overline{S_{TD}}$) and the software productivity *Prd* in terms of kLOC/Person-Year.

$$IWL = \frac{\overline{S_{PrJ}}}{Prd} \text{ [Person-Month]} \quad (4)$$

- **(e) Computing Optimal Labor Allocation:** The optimal labor allocation LR_0 of a software engineering project for a given ideal workload W_1 is obtained by the differentiable function *dTS/dLR* based on Eq. (1) when its derivative equals to zero.

$$\frac{dTS}{dLR} = \frac{d}{dLR}\left(\frac{1}{2} IWL\left(cr * LR - cr + \frac{2}{LR}\right) = \frac{1}{2} IWL(cr - \frac{2}{LR^2}) = 0\right) \quad (5\text{-}1)$$

[Person]

So, the optimal labor allocation LR_0 of a software engineering project is determined by solving the Eq. (5-1) (Wang, 2006a). Since the W_1 is not zero, the term cr $-\frac{2}{LR^2} = 0$ yields to the following result that is the answer for the optimum labor allocation.

$$LR_0 = \frac{1.414}{\sqrt{cr}}\ , cr \neq o \ \text{[Person]} \qquad (5\text{-}2)$$

- **(f) Computing Shortest Duration:** In the software industry, time to market is always a priority. Therefore, the shortest duration optimization strategy is as practically important as that of the cost optimization strategy. The shortest duration TSmin of software engineering projects under the optimum labor allocation L_o for a given ideal workload W_1 is as follows (24):

$$TS_{min} = \{TS|LR = LR_0\} = \frac{1}{2}(IWL\left(cr * LR_0 - cr + \frac{2}{LR_0}\right) \text{ [Month]} \qquad (6)$$

- **(g) Computing Minimum Effort:** The minimum workload for a software project is W_{min} = IWL. The strategy for the optimization of a software engineering project for the lowest effort is to set the project at WL(TS_{min}, LR_o). Otherwise, the waste of effort ΔWL can be determined by Eq.(7-1) [(Pressman, 2014), (Rashidi, 2014), (Wang, 2006a)]:

$$\Delta WL = WL - WL_{min} \quad \text{[Person-Month]} \qquad (7\text{-}1)$$

where WL is the real workload due to non-optimal work allocation. When the optimal labor allocation and the shortest duration of the project are determined, the optimal real effort or workload of the project is obtained by Eq.(7-2):

$$WL_{min} = LR_0 \times TS_{min} \quad \text{[Person-Month]} \qquad (7\text{-}2)$$

- **(h) Optimize Project Costs:** Based on the minimized project workload, the cost of the given project can be determined. The estimated cost of a software project CP is a product of the optimal real workload WL_{min} [PM] and the average cost of labor AC_L [$/PM], as calculated by Eq. (8) [(Pressman, 2014), (Rashidi, 2014), (Wang, 2006a)]:

$$CP = WL_{min} \times AC_L = LR_0 \times TS_{min} \times AC_L \quad \text{[Person-Month]} \qquad (8)$$

It is noteworthy, the cost CP determined by Eq. (8) covers only the operational cost in the economic term. There are additional costs, mainly capital costs such as office, facilities, and developing environment, in economic analysis. A complete economic analysis of software engineering projects may be referred to (Wang, 2006b).

4. Results

In this section, the results of executing the **methodology in this research, are** presented. The flowchart of the implementation is illustrated in Figure 5. The flowchart of this figure is implemented by PlannersLab (Wagner, 2018), which is a development tool for decision support systems.

The dashboard in the subsystem of the user interface (see Fig. 1) is used to maintain/manipulate temporary values in order to obtain a convergence between the size of the project in top-down and bottom-up approaches, i.e. the values obtained from Eq. (1) and Eq. (2).

Once the size of a given project is obtained, the workload WL can be accurately determined on the basis of software productivity benchmarks of the whole industry or the historical data of a specific organization. Based on this, decision optimization for optimal labor allocation can be carried out. The strategy for optimizing a project for the shortest duration is to set the project at WL(TR_{min}, LR_O), where LR_o is the optimal labor allocation for a given project, and TR_{min} is the corresponding shortest duration of the project with LR_o.

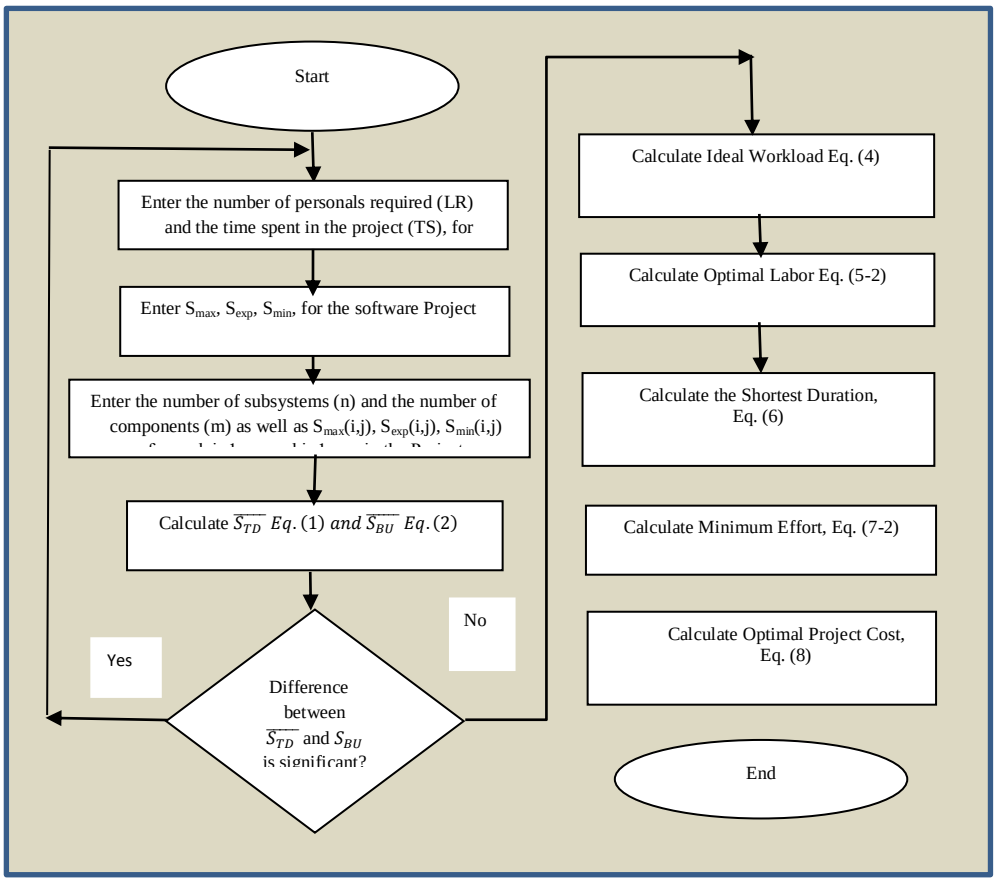

Fig. 5: Flowchart of the evaluation the building block of the model subsystem

We investigated the proposed system on NASA60 dataset in which there are 60 projects (Menzies et al., 2005). In this dataset, for each project, thousands of line code (KLOC), Actual Efforts (ACT_EFFORT) and several features are given. Moreover, Table 2 shows the attributes of the project in this dataset. The columns 1-3 of the table show a description of the feature, the abbreviation used, and possible values, respectively, in the dataset. The value of each feature in every project is given in the dataset.

Table 3 shows an evaluation of the equations in the core model subsystem of the DSSSCE. The columns 1 to 3 show the data from the dataset. If we want to run the Eq. (4) in the DSSSCE, we found a typical benchmark of *Prd* is 3,000 LOC per year when management, quality assurance, and supporting activities are considered (See (Boehm, 1987), (Brooks, 1975), (Jones, 1981), (Jones, 1986), (Livermore, 2005), (Wang, 2006a). The columns (b) and (c) of the table shows the result of IWL for Prd=3000 LOC/Year by calculating Eq. (4) and DWL by calculating Eq. (7-1). The column (d) is the difference percentage of the columns (a) and (b).

Table 2 - The attributes of the projects used in NASA60 dataset

Feature	Abbreviation	Possible Values
Analysts Capability	ACAP	Nominal, High, Very_High
Programmers Capability	PCAP	Nominal, High, Very_High
Application Experience	AEXP	Nominal, Very_High, High
Modern Programming Practices	MODP	Low, Nominal, High, Very_High
Use Of Software Tools	TOOL	Very_Low, Low, Nominal, High, Very_High,
Virtual Machine Experience	VEXP	Low, Nominal, High
Language Experience	LEXP	Very_Low, Low, Nominal, High
Schedule Constraint	SCED	Low, Nominal, High
Main Memory Constraint	STOR	Nominal, High, Very_High, Extra_High
Data Base Size	DATA	Low, Nominal, High, Very_High
Time Constraint For CPU	TIME	Nominal, High, Very_High, Extra_High
Turnaround Time	TURN	Nominal, Low, High
Machine Volatility	VIRT	Low, Nominal, High
Process Complexity	CPLX	Low, Nominal, High, Very_High, Extra_High
Required Software Reliability	RELY	Low, High, Nominal, Very_High

Table 3 - An evaluation of the equations in the core model subsystem of the DSSSCE

Project	KLOC in Project	(a) ACT_EFFORT in Project	(b) IWL for Prd=3000 LOC/Year by Eq. (4)	(c) DWL by Eq. (7-1)	(d) Difference Percentage [(a)-(b)]/(a) *100
1	70	278	280	-2.0	-0.72
2	227	1181	908	273.0	23.12
3	177.9	1248	711.6	536.4	42.98
4	115.8	480	463.2	16.8	3.50
5	29.5	120	118	2.0	1.67
6	19.7	60	78.8	-18.8	-31.33
7	66.6	300	266.4	33.6	11.20
8	5.5	18	22	-4.0	-22.22
9	10.4	50	41.6	8.4	16.80
10	14	60	56	4.0	6.67
11	16	114	64	50.0	43.86
12	6.5	42	26	16.0	38.10
13	13	60	52	8.0	13.33
14	8	42	32	10.0	23.81
15	90	450	360	90.0	20.00
16	15	90	60	30.0	33.33
17	38	210	152	58.0	27.62
18	10	48	40	8.0	16.67
19	161.1	815	644.4	170.6	20.93
20	48.5	239	194	45.0	18.83
21	32.6	170	130.4	39.6	23.29
22	12.8	62	51.2	10.8	17.42
23	15.4	70	61.6	8.4	12.00
24	16.3	82	65.2	16.8	20.49
25	35.5	192	142	50.0	26.04
26	25.9	117.6	103.6	14.0	11.90
27	24.6	117.6	98.4	19.2	16.33

Project	KLOC in Project	(a) ACT_EFFORT in Project	(b) IWL for Prd=3000 LOC/Year by Eq. (4)	(c) DWL by Eq. (7-1)	(d) Difference Percentage [(a)-(b)]/(a) *100
28	7.7	31.2	30.8	0.4	1.28
29	9.7	25.2	38.8	-13.6	-53.97
30	2.2	8.4	8.8	-0.4	-4.76
31	3.5	10.8	14	-3.2	-29.63
32	8.2	36	32.8	3.2	8.89
33	66.6	352.8	266.4	86.4	24.49
34	150	324	600	-276.0	-85.19
35	100	360	400	-40.0	-11.11
36	100	215	400	-185.0	-86.05
37	100	360	400	-40.0	-11.11
38	15	48	60	-12.0	-25.00
39	32.5	60	130	-70.0	-116.67
40	31.5	60	126	-66.0	-110.00
41	6	24	24	0.0	0.00
42	11.3	36	45.2	-9.2	-25.56
43	20	72	80	-8.0	-11.11
44	20	48	80	-32.0	-66.67
45	7.5	72	30	42.0	58.33
46	302	2400	1208	1192.0	49.67
47	370	3240	1480	1760.0	54.32
48	219	2120	876	1244.0	58.68
49	50	370	200	170.0	45.95
50	101	750	404	346.0	46.13
51	190	420	760	-340.0	-80.95
52	47.5	252	190	62.0	24.60
53	21	107	84	23.0	21.50
54	423	2300	1692	608.0	26.43
55	79	400	316	84.0	21.00
56	284.7	973	1138.8	-165.8	-17.04
57	282.1	1368	1128.4	239.6	17.51
58	78	571.4	312	259.4	45.40
59	11.4	98.8	45.6	53.2	53.85
60	19.3	155	77.2	77.8	50.19

5. Discussion and Conclusions

In this section, the major discussions over the results of executing the methodology and conclusion are presented. The discussions are in the form of following corollaries that obtained from the equations of the model building block used in the methodology:

- **Corollary-1**: The results show that if some attributes of the software project increases, they decrease the actual and ideal efforts. These attributes are ACAP, PCAP, AEXP, MODP, TOOL, VEXP, and LEXP.
- **Corollary-2:** If some attributes of the software project decrease, they decrease the actual and ideal efforts. These attributes are STOR, DATA, TIME, TURN, VIRT, CPLX and RELY.
- **Corollary**-3: The results show that the attribute SCED has no significant effect on increasing or decreasing the actual and Ideal workload.
- **Corollary-4**: Eq. (5-2) reveals that the optimal labor allocation, LR_0, for a given project is solely determined by the rate of coordination r. In software engineering projects, the coordination rate is usually within the scope of $1\% < rc < 50\%$.

This information helps us to obtain the scope of the optimal labor allocation for any software engineering project as Eq. (9) (Wang, 2006a):

$$1 \leq LR_0 \leq 14 \text{ [Person]} \quad (9)$$

- **Corollary-5**: Eq. (5-1) and (5-2) reveal it is noteworthy that the labor allocation is not directly determined by project size. That is, it is interesting that no matter how large a software project is, the optimum labor allocations are mainly ranged within one through ten persons. Any other solutions are not an optimum labor allocation, because they do not result in the shortest project duration, rather than create a dramatically large actual workload.
- **Corollary-6**: If we make a constraint on a group size of the development team in collaborative work with an upper limit of group size GS_{max} at 20, as Eq. (10):

$$GS_{max} = \max(LR_0(cr)) = 20 \text{ [Person]} \quad (10)$$

We need a coordination rate cr = 0.005.

- **Corollary-7**: To satisfy $LR_0 > 20$ Person, there must be a coordination rate cr < 0.005. Because a coordination rate *cr* less than 0.5% is impossible for many software engineering projects [(Wang, 2006a), (Wang, 2006b), (Wood, and Gray, 1991)], therefore we don't recommend the labor required in the software project with more than 20. Otherwise, software projects may not be organized economically, efficiently, and technically sound in any form.
- **Corollary-8**: Any effort to violate the constraint specified by Eq. (10)-adding more labor into a maximum labor allocated project- will result in an exponentially increased actual workload, or in other words, a project failure in usual. This may contradict to some empirical intuitions. However, the reality has been proven by so many failed projects in software engineering that involve hundreds of programmers in a single project (see (Wang, 2006a), (Wang, 2006b), (Wood, and Gray, 1991)).
- **Corollary-9**: the results show that when we have a large collaborative engineering project with the ideal workload (IWL) to 100 Person-Month where there is a higher coordination rate *cr*, we cannot complete the project economically and feasibly in less than TS_{min} = 5 Month.
- **Corollary-10**: The results show that setting TS_{min} to 10 Month (less than one year) is usually safe for IWL of 100 Person-Month. It is the same as suggested in (Lunesu et al., 2018). Of course, there are some managers who still attempt to organize large software engineering projects that require a shorter duration than 5 months with more than 20 persons. The only possible clue to do so is to divide the whole system into clearly partitioned and isolated parallel subsystems, subject to that each of those subsystems should still obey the constraint on group sizes in collaborative work.

This paper presents a model-driven Decision Support System for Software Cost Estimating (DSSSCE). The system consists of four subsystems, namely Data subsystem, Model subsystem, User Interface subsystem, and Knowledge subsystem. The core of DSSSCE supports this system is based on coherent theory on the nature of collaborative work and their mathematical models in software engineering that included in the model subsystems. This core provides a theoretical foundation for software engineering decision optimizations on the optimal labor allocation, the shortest duration determination, and the lowest workload effort and costs estimation.

This paper has systematically presented a rigorous treatment of building block in the model subsystem. DSSSCE is adopted in analyzing the cost factors and their relations. A set of theories in software engineering costs estimation and optimization has been derived that enable the formal analyses of software engineering project organizations. The experimental results and evaluations on Dataset NASA60 showed that the proposed system has significant conformance with experience in practice. The mathematical equations in the DSSSCE indicate that the strategy for the optimization of a software engineering project for the lowest cost is to set the project at $WL(TS_{min}, LR_O)$, where LR_O is the optimal labor allocation for a given project, and TSR_{min} is the corresponding shortest duration of the project with LR_O.

This research provides a mathematical foundation for software cost estimation in the form of a Decision Support System. The main result of the evaluation revealed that software project cost is more directly

related to the expected workload of the project. Based on the mathematical equations in the DSSSCE, a wide range of applications in optimal software engineering organizations can be studied. For the accountant and financial analyst in software companies, DSSSCE provides some attractive features for the formulation of budgets, budgetary planning and control, and financial analyses. The initial estimation is an important factor for project management because it reduces the margin of error in estimating software cost. This is particularly true when relatively large software components such as subsystems are reused.

References

1) AIS Special Interest Groups (2018), Association For Information Systems, https://aisnet.org/, Last Date Visit: 3 Oct 2018
2) Alter, S.L. Decision Support Systems: Current Practice and Continuing Challenge. Reading, MA: Addison-Wesley, 1980.
3) Bhandari S. (2016), FCM based conceptual framework for software effort estimation, 3rd IEEE International Conference on Computing for Sustainable Global Development (INDIACom), pp. 2584-2588.
4) Boehm B.W. (1987), Improving Software Productivity, IEEE Computer, Vol. 20, No. 9, pp.43.
5) Brooks F.P. Jr. (1975), The Mythical Man-Month. Essays on Software Engineering, Addison Wesley Longman, Inc., Boston.
6) Burstein F.; Holsapple C. W., Handbook on Decision Support Systems, Berlin: Springer Verlag, 2008
7) Daniel J. Power, Decision Support Systems: Concepts and Resources for Managers, Greenwood Publishing Group, 2002
8) Donzelli, P. (2006), A Decision Support System for Software Project Management. IEEE Software, Vol. 23(4): p. 67-75.
9) Holsapple, C.W., and A. B. Whinston (1996), Decision Support Systems: A Knowledge-Based Approach, West Publishing.
10) Jones C. (1981), Programming Productivity – Issues for the Eighties, IEEE Press, Silver Spring, MD.
11) Jones C. (1986), Programming Productivity, McGraw-Hill Book Co., NY.
12) Kan Qi ; Barry W. Boehm (2017), A light-weight incremental effort estimation model for use case driven projects, 28th IEEE Annual Software Technology Conference (STC), pp. 1-8.
13) Li Jun L., Jianming L., Yongqin1 J., Qingzhang C. (2008), Development of the Decision Support System for Software Project Cost Estimation, 2008 International Symposium on Information Science and Engineering.
14) Livermore J. (2005), Measuring Programmer Productivity, http://home.sprynet.com/-jgarriso/
15) Lunesu M.L., Münchb J., Marchesic M., Kuhrmann M. (2018), Using simulation for understanding and reproducing distributed software development processes in the cloud, Information and Software Technology, Vol. 103, PP. 226-238
16) Menzies. T, Port. D, Chen. Zh, Hihn. J. (2005), Validation Methods for Calibrating Software Effort Models, ICSE ACM.
17) Pashaei Barbin J., Rashidi H. (2015), A Decision Support System for Estimating Cost of Software Projects Using A Hybrid Of Multi-Layer Artificial Neural Network and Decision Tree, International Journal In Foundations Of Computer Science & Technology Vol.5 (6), PP 23-31.
18) Pressman R. S. (2014), Software Engineering: A Practitioner's Approach, 8th Edition, McGraw-Hill.
19) Rashidi H. (2014), Software Engineering-A programming approach," 2nd edition., Allameh Tabataba'i University Press (in Persian), Iran.
20) Sangwan, Om Prakash (2017). Software effort estimation using machine learning techniques, Cloud Computing, Data Science & Engineering-Confluence, 2017 7th International Conference on. IEEE.
21) Sharda R ., Aronson J., Turban E. (2015), Business Intelligence and Analytics: Systems for Decision Support, 10th Edition, Pearson Prentice Hall.
22) Sommerville Y. (2018), Software Engineering, 10th Edition, Pearson Education.
23) Wagner J., Planners Lab Software, TeraData University Network, http://plannerslab.com/, Last Date Visit: 3 Oct 2018
24) Wang Y. (2006a), A Mathematical Model for Explaining the Mythic Man-Month, Proceedings of the 19th IEEE Canadian Conference on

Electrical and Computer Engineering (CCECE'06), Ottawa, Canada, May.
25) Wang Y. (2006b), Software Engineering Foundations. A Transdisciplinary and Rigorous Perspective, CRC Software Engineering Series, Vol.2, CRC Press, USA.
26) Wood D. J. and B. Gray (1991), Towards a Comprehensive Theory of Collaboration, Journal of Applied Behavioral Science, 27(2), 139-162.
27) Venkataiah, V., Mohanty, R., Nagaratna, M. (2019), Application of Hybrid Techniques to Forecasting Accurate Software Cost Estimation, International Journal of Recent Technology and Engineering 7(6), pp. 408-412.
28) El Bajta, M., Idri, A. (2019), A Software Cost Estimation Taxonomy for Global Software Development Projects, ICSOFT 2019 - Proceedings of the 14th International Conference on Software Technologies, pp. 218-225.
29) Resmi, V., Vijayalakshmi, S. (2019), Kernel Fuzzy Clustering With Output Layer Self-Connection Recurrent Neural Networks for Software Cost Estimation, Journal of Circuits, Systems and Computers, Article in press.
30) Rivadeh, M., Khadivar, A. (2019), A model for software development cost Estimation with System Dynamic Approach, Iranian Journal of Information Processing Management 34(3), pp. 1343-1370.
31) Parthasarathi Patra, H., Rajnish, K. (2019), A New High-Performance Empirical Model for Software Cost Estimation, International Journal of Computer-Aided Engineering and Technology 11(4-5), pp. 601-612.
32) Naik, P., Nayak, S. (2019), Intelligence-Software Cost Estimation Model for Optimizing Project Management, Advances in Intelligent Systems and Computing 984, pp. 433-443.
33) Khan, M.S., Ul Hassan, C.A., Shah, M.A., Shamim (2018), A., Software Cost and Effort Estimation Using a New Optimization Algorithm Inspired by Strawberry Plant, ICAC 2018 - 2018 24th IEEE International Conference on Automation and Computing: Improving Productivity through Automation and Computing, 8749003.
34) Prasad, B.M.G., Sreenivas, P.V.S., Veena, C., Software Project Effort Duration and Cost Estimation Using Regression Testing and Adaptive Firefly Algorithm (AFA) (2019), International Journal of Engineering and Advanced Technology 8(2), pp. 104-111