Hong Kong International Journal of Research Studies, ISSN: 3078-4018
Volume 4, Issue 1, January-June, 2026
Available online at: https://octopuspublication.com/index.php/hkijrs

A Study on "Key Activities in Managing Software Project Effort"

Godaparthi Kalpana

Lecturer in Computer Science, Govt. Degree College (A)-Bhadrachalam, Bhadradri Kothagudem District, Telangana, India

ABSTRACT

Accurate effort estimation is a crucial task for software business progression: for customers, acquiring software products or making project contracts for software implementation, accurate effort estimation enables adherence to schedule and budget without delay in deployment and introduction (Conte, et al., 1986; Sommer- ville, 2001). A software supplier organization strives to estimate the effort needed in building software as accurately as possible to ensure the project's budget and schedule, and the success of resource allocation. Despite the numerous effort estimation approaches and applications available, the estimates have remained inaccurate. The objective of this paper is to find out the management practices of software development project effort, resulting in increased effort estimate accuracy. In the quest of its goal, the paper commences by presenting the theoretical background and the key concepts related to software project effort management approach. This paper focuses on an activity set of general software project activities, which are found to be one of the major software project activity categories besides software construction and project management effort.

Keywords: project management, software development effort management

INTRODUCTION

Software providers require cost, price, and time-to-market calculations which are not possible without knowledge of effort and its distribution. Workload estimation is needed for work and staffing plans. Sufficiently accurate effort estimation is important for software engineers, because successful resource allocation decreases working pressure and haste. Accurate effort estimation is a crucial task for software business progression: for customers, acquiring software products or making project contracts for tailored software implementation, accurate effort estimation enables adherence to schedule and budget without delay in deployment and introduction (Conte, et al., 1986; Sommerville, 2001). The numerous formal models, methods and tools available for estimation have reached accuracy levels which are too low for the software industry companies. In practice, the same level of accuracy can be achieved with the informal expert judgment technique, which has remained as the most commonly employed estimation technique (Jørgensen, 2007). Two factors are emphasized in respect to the difficulty in producing accurate estimates: software sizing and the available data for estimations (Armour, 2002). Moreover, it has been argued that the formal effort estimation models are too complex and uncertain for practical use (Sommerville, 2001). The applications of these models are usually not transparent, i.e., the factor weights used for effort derivation are not obtainable in order to validate them. Transparency would be required by the estimator to evaluate the reasonability of the gained estimate, i.e., what comprises the effort and the costs.

An important, yet overlooked, factor explaining poor estimates is the light consideration on different activities involved with the project. The estimators employing the expert judgment technique frequently consider only those activities that they are in relation with in a software project and focus on the effort of actual software construction (Jørgensen, 2004b, 2005; Jørgensen and Sjøberg, 2004). Moreover, the focus in effort estimation research and estimation applications has been on software construction (i.e., analysis, design, implementation and testing) and project management (Boehm, et al., 2000; MacDonell and Shepperd, 2003), whilst neglecting other software project activities.

LITERATURE REVIEW

The theoretical background for the key activities related in managing effort in a software project is introduced in this section.

EFFORT AND COST ESTIMATION

The *effort* of a software development project can be generally defined as the time consumed by the project, and it can be expressed as a number of person hours, days, months or years, depending on the size of the project (Chatters, et al., 1999). Brooks (1975) has defined effort as the product of people and time, i.e., effort = people * time. Effort is estimated in most projects, especially in projects employing traditional project management methods, to derive the

project costs that are needed to justify the business case. In agile software development, the project costs are derived from the estimated software size (Cohn, 2008). An *estimate* is a probabilistic assessment with a reasonable accurate value of the centre of a range. Formally, an estimate is defined as the median of the (unknown) distribution (Fenton and Pfleeger, 1997). An estimate is a prediction; hence, an estimation model can be considered as a prediction system. *Effort estimation* is often used as a synonym for cost estimation but, to be precise, *cost estimation* is derived from the effort estimate by valuing the effort using the cost of project personnel and some other major project costs. Thus the outcome of effort estimation is a time value, whereas the cost estimate outcome is some monetary value (Conte, et al., 1986; Fenton and Pfleeger, 1997; Sommerville, 2001).

EFFORT DISTRIBUTION

Although effort can be distributed in a project between project activities or project phases in many different ways, *effort distribution* is a less-investigated effort estimation area. Indeed, it has been disputed whether it is useful to distribute effort in the first place (Blackburn, et al., 1996; MacDonell and Shepperd, 2003). Yet examples exist in the software engineering literature; Brooks' (1975) suggestion for rule-of-thumb effort distribution for software construction among the first in the mid-1970s. Effort distribution can also be used to compare different projects with each other (Heijstek and Chaudron, 2008).

WORK BREAKDOWN STRUCTURE (WBS)

Effort is distributed on software project activities and sets of activities that form a *work breakdown structure*, WBS. A WBS is a particular defined tree-structure hierarchy of elements that decomposes the project into discrete work tasks (Royce, 1998; Wilson and Sifer, 1988). It can be very useful to organize project activity elements into a hierarchical structure for project budgetary planning and control purposes (Boehm, 1981). An early establishment of a WBS helps to divide the effort into distinct work segments that can be scheduled and prioritized (Agarwal, et al., 2001).

WBS is also required by the currently employed capability maturity models. For example, the staged representation of CMMI (SEI, 2006; SEI, 2010) requires WBS on its maturity level 2. However, no standardized way to create a WBS exists, and the software engineering literature provides only a few general WBS including ones applied with the two COCOMO models (Boehm, 1981; Boehm, et al., 2000), the *Rational Unified Process* (RUP) activity distribution (Royce, 1998), and the ISO/IEC 12207 work breakdown structure for software lifecycle processes (ISO, 1995). It is noted that although the concept and practice of using a WBS is well established, the topic is largely avoided in the published literature because the development of a WBS depends on the project management style, organizational culture, customer preference, financial constraints, and several other project-specific parameters (Royce, 1998).

Another structure for work breakdown is provided in the OPEN process Framework (Henderson-Sellers, 2003), which in one of the five major framework compo- nents is Work Units, which results as a Work Product. A *Work unit* is defined as a functionally cohesive operation performed by a Producer (people). The three major classes of Work Unit are Activity (e.g., 'project planning' or 'modeling and implementation'), Task (e.g., 'code' or 'evaluate quality'), and Technique (e.g., 'class naming 'prototyping' or 'unit testing'). These classes form pairs and are linked with each other, e.g., Activities are linked with Tasks, and Tasks with Techniques.

EFFORT MANAGEMENT

The concept of effort management is rarely used in software engineering or information system research. The concept was introduced in information system research first in 1995. Young (1995, p. 716) defines *effort management* as a management area that "tracks the commitment of resources against undertakings, both project and non-project". In fact, Young (1995) emphasizes the non-project consideration over project. We, on the other hand, implicitly limit our consideration to software projects, and define effort management as an organized process to estimate, collect, monitor and control, and analyse effort related to software projects and their activities.

Effort management can be considered to include all necessary functions which manage effort in a software project. These functions include effort estimation, effort collection, and effort assessment and analysis, for instance. Simplified, during project planning, effort is initially estimated with the method in use. The estimate is revised during project execution with both collected and assessed effort data from the project. During project closure, the delivered project and its effort are analysed, and a project final report is drawn. To assist effort management, some frameworks have been proposed: *Software Development Management* (SDM) framework (Tsoi, 1999), and the frameworks proposed in (Fairley, 1992; Vesterinen, 1998). These frameworks are fairly limited, both in terms of project lifecycle phases and effort functions. For example, the SDM framework is limited mostly to the pre- project and project phases. The proposed frameworks also concentrate mostly on the effort estimation function.

EFFORT ESTIMATIONS AND RE-ESTIMATIONS

The effort of a software project can be estimated and modelled, and estimation techniques can be classified in several ways. In the following, we present the evolution from the first effort and cost estimation approach classifications by

Boehm (1981) and Conte, et al., (1986) to the later classifications by Fairley (1992), Fenton and Pfleeger (1997) and Briand, et al., (1999, 2000).

Boehm (1981) divided the methods into seven categories: algorithmic models, expert judgment, analogy, Parkinson, price- to-win, top-down, and bottom-up. The algorithmic models include linear models, multiplication models, analytic models, tabular models, and composite models. Later, this division was reduced to two main antithesic approaches: algorithmic and non-algorithmic approaches. Conte, et al., (1986) divided the approaches of effort estimation on the highest level into micro-model and macro-models of effort. The macro-models of effort were divided further into historical-experimental models, statistically-based models, theoretically-based models, and composite models. The historical-experimental models include expert judgment technique. The statistically-based models include techniques such as regression analysis, linear models, and non-linear models.

In 1992, Fairley (1992) addressed the different estimation techniques which included the traditional approaches (categorized into empirical techniques, regression techniques, and theory-based techniques) and the advances in the popular approaches at that time (advances in analogy-based estimation, function point tech- niques, regression modelling, theory-based models, and in size estimation). Fairley (1992) predicted that analogy-based methods with expert system decision rules will emerge into future estimations.

Fenton and Pfleeger's (1997) division was based much on Boehm's division, but included only four categories: expert opinion, analogy, decomposition, and models. Each of these techniques can be applied either bottom-up or top-down.

Briand, et al., (1999, 2000) divided the effort estimation techniques into parametric and more advanced non-parametric modelling techniques (Figure 1). The non- parametric modelling techniques have emerged gradually into effort estimation, but applications are still in short supply.

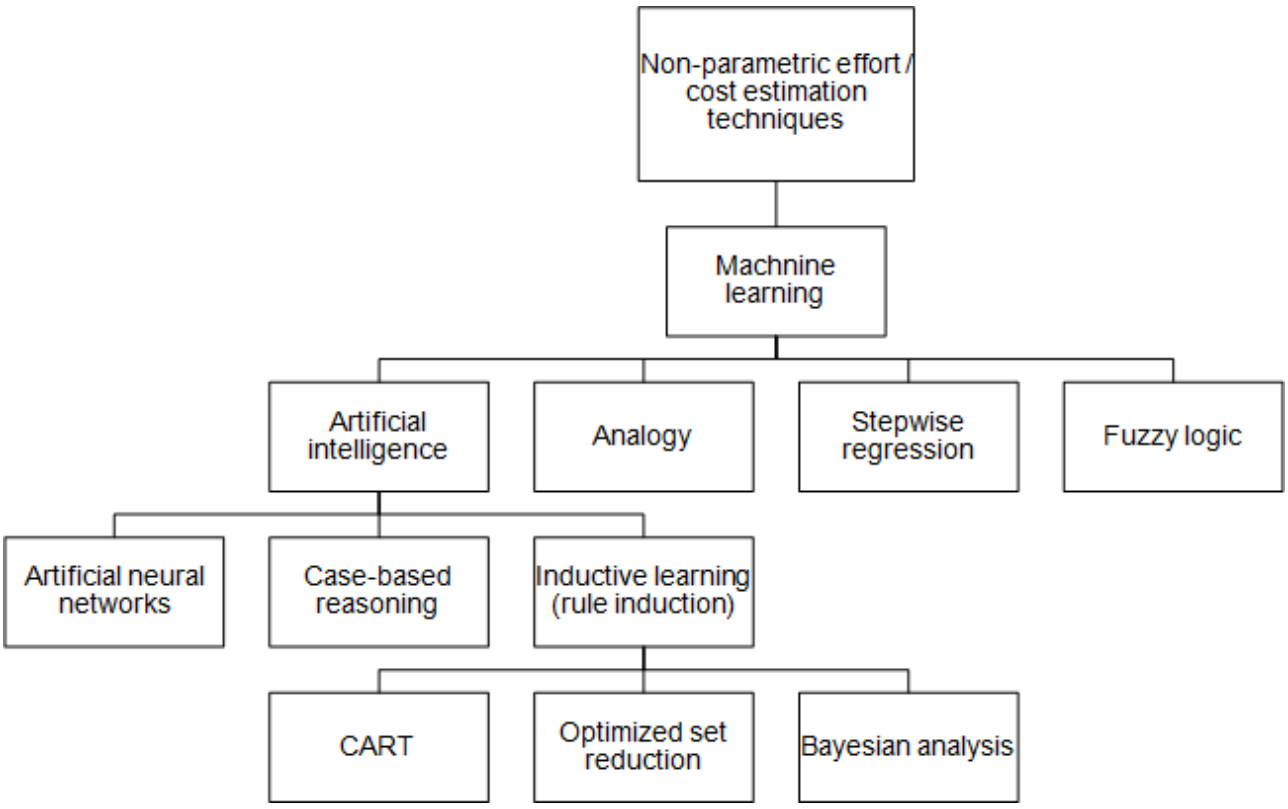

Figure 1: The Non-Parametric Effort Estimation Technique Classification

(adopted from Briand, et al., 1999, 2000)

Since the beginning of the 1990's research on effort estimation has increased and new approaches have been proposed and presented. These approaches include various machine learning techniques and artificial intelligence. Based on the evaluations of these advanced techniques (e.g., (Finnie and Wittig, 1996; Mukhopadh- yay and Kekre, 1992; Shepperd and Schofield, 1997; Srinivasan and Fisher, 1995)), one cannot conclude a superior estimation technique, although case-based reasoning seems to be a promising one out of the advanced formal approaches. The results of the evaluations depend on the case and are partially in conflict with each other. Although new approaches have been introduced, in

practice the methods employed are mostly old approaches. A reason for this is that the new approaches, such as artificial neural networks, produce results hard to interpret in practice (Briand, et al., 1999).

In fact, formal techniques in general are seldom employed in the software engineering industry. Studies (Gray, et al., 1999; Hihn and Habib-agahi, 1991; Jørgensen, 2005; Jørgensen and Sjøberg, 2004; Kitchenham, et al., 2002; Virtanen, 2003) imply that applied techniques in the software industry are primarily informal, and based on analogy and expert analysis, particularly since there is no conclusive evidence that a formal method outperforms the informal expert analysis (Jørgensen, 2004a, 2004b, 2007; Kitchenham, et al., 2009).

In another study (Niessink and van Vliet, 1997), the expert judgment technique has been reported to outperform the Function Point Analysis based estimations in the context of predicting maintenance effort. *Expert judgment estimation* is defined as an "estimation conducted by a person who is recognized as an expert on the task and who follows a process that is, to some degree, non-explicit and non-recoverable" (Jørgensen and Sjøberg, 2004, p. 317). Reasons for applying expert judgment instead of formal estimation methods include flexibility regarding required input for estimations and time spent on estimations.

Software project effort estimation has been increasingly studied since 1960's both in the academia and in the software industry. In particular, the research interest lies within *modelling* effort and cost estimation. As models in general, they describe a phenomenon *ex post*, and thus their use *ex ante* for estimation can be challenging; rather, they can be used for simulating the software project effort after project's completion when the actual effort is in hand. Moreover, the models require information as input parameters which cannot be obtained before the project is completed. For example, COCOMO model (Constructive Cost Model)
(Boehm, 1981) requires the number of lines of code when modelling software project effort.

Effort and cost estimation has existed in software engineering research literature since 1960's when the first models to estimate effort and cost were introduced (Conte, et al., 1986). These pioneering models included linear statistical models such as Nelson and Farr-Zagorsky. The famous formal models are from the turn of the 1970-80's, and include the Putnam SLIM model, Price-S model, the Jensen model, and the COCOMO model.

According to (Pressman, 2005; Royce, 1998; Smith, et al., 2001), the most widely known models include the COCOMO (Constructive Cost Model) (Boehm, 1981) and COCOMO II (Boehm, et al., 1995, 2000) models, which are primarily based on software size in lines of code, and the *Function point* (FP) count (Albrecht and Gaffney, 1983). The FP count, or *Function Point Analysis* (FPA), is based on the software functionality, and can be used to derive effort with a known productivity factor, and has many variants such as Mark II FP (Symons, 1988).

Boehm (1981) introduced the original COCOMO model in 1981. The original *COCOMO model* is a collection of three different models: The Basic model, the Inter- mediate model, and the Detailed model. The purpose of using a more complex model is to achieve more accurate estimation by taking more factors into account. This in turn requires more details to be known. The Basic model, which can be applied in the early stage of the project, estimates the effort based primarily on the software project's size in terms of program lines of code (*kilo delivered source instructions*, KDSI). The Intermediate and Detailed models use an *Effort Adjustment Factor* (EAF) and slightly different coefficients for the effort equations. The EAF is a product of the six-scaled Effort Multiplier and the corresponding cost driver. The COCOMO cost drivers consists of fifteen independent Product, computer, personnel and project attributes which determine the project's effort. The major difference between the Intermediate and the Detailed model is that Effort Multipliers are in the Detailed model used for each project phase (Agarwal, et al., 2001; Boehm, 1981; Fenton and Pfleeger, 1997). COCOMO was enhanced to *COCOMO II* in the mid-1990's in order to develop the model to address the new needs of evolving software engineering such as distributed software and component techniques (Boehm, et al., 2000; Fenton and Pfleeger, 1997). COCOMO II categorization is not, however, suitable for all project situations, and should be adjusted via context and judgment to fit individual projects (Boehm, et al., 2000). COCOMO II model's equation for counting effort corresponds with the original COCOMO's Intermediate (or Detailed) model's equation. The amount of Effort Multipliers considered depends on the project's development phase as the knowledge on the project grows (Agarwal, et al., 2001). Moreover, besides using only lines of code for the software sizing, object or function points can be used (Fenton and Pfleeger, 1997; Pressman, 2005).

Besides estimating effort and cost, the COCOMO models provide effort distributions over different work breakdown structures. A WBS was suggested for projects employing a waterfall project lifecycle process in (Boehm, 1981). The WBS is applied with the COCOMO cost estimation model as the model estimates how effort is distributed on different activities between eight major categories, namely requirement analysis, product design, programming, test planning, verification and validation, project office functions, configuration management and quality assurance, and manuals, each having specific activities in the four project lifecycle phases. These phases were based on earlier estimation methods and the waterfall lifecycle model (Boehm, 1981).

COCOMO II has been developed to be usable by projects employing either waterfall or spiral software engineering processes. In the waterfall approach of COCOMO II, software activity work was divided into five major categories: management, system engineering, programming, test and evaluation, and data. This breakdown was adapted from the COCOMO's eight categories, which were partly reorganized and renamed. The requirements, product design, and configuration manage- ment and quality assurance categories were organized as system engineering sub-activities. 'Verification and validation' was renamed 'test and evaluation', 'man- uals' became 'data', and 'project office functions' became 'management' (Boehm, et al., 2000).

The COCOMO II spiral approach is based on the same WBS approach applied in the RUP (Royce, 1998) The RUP default WBS is based on seven main categories, namely management, environment, requirements, design, implementation, assessment, and deployment. Each of these seven categories includes activities relat-ing to four project phases (inception and elaboration of the engineering stage, and construction and transition of the construction stage) (Royce, 1998). The COCOMO II WBS for spiral process is based much on the same seven categories, yet RUP's 'environment' became 'environment and configuration management' in COCOMO II. Also, the activities within the four phases partly differ (Boehm, et al., 2000). Moreover, COCOCO II dropped COCOMO's modern- programming-practices parameter in favour of a more general process-maturity parameter. COCOMO II was also enhanced with several new parameters (Chen, et al., 2005), for example, with the development of reuse, multi-site development, architecture and risk resolution, and team cohesion. Moreover, the organization can add new proprietary parameters reflecting its particular situations.

EFFORT DATA COLLECTION

The software engineering literature (e.g., Pressman, 2005; Royce, 1998; Sommerville, 2001) describes effort-related functions during project execution (collection, monitoring, assessments, and re-estimations) very superficially compared to the effort-related functions during project planning (e.g., effort estimation, scheduling etc.).

However, a few descriptions exist. In the beginning of the 1980s, Boehm (1981) described a follow-up system for projects' effort (cost) information. He pointed out that in the beginning effort estimation is imperfect and actual effort data is needed for calibration and change management. Moreover, not every project fits into the estimating model. Hence, the employed formal effort estimation model requires a particular follow-up system. This system both supports effective project management and benefits the long-range effort estimation capabilities. The data collected in a consistent manner via control activities over several projects can be analysed to determine how the actual effort distribution differs from the estimates. The differences are fed back to calibrate the model. For a new project, data collection should be considered to calibrate their estimating models (Boehm, 1981).

The importance of the effort-related functions during project execution, such as effort collection, was raised also in (Tsoi, 1999), where a framework for software project development management was proposed. This framework concentrated on the pre-project and project phases (referred as acquisition and operation phases, respectively), and especially on the monitoring the effort in a software project. The framework relied on two basic principles which occur in software projects: the existence of change due to the dynamic environment, and the need for dynamic, continuous measurement on project progress to collect real-time information.

RESEARCH METHODOLOGY

Several different research strategies have been proposed for the software engineering and information systems research. These research methodologies can be found very useful, but no-one is sufficient by itself to form a well-grounded research program. In fact, Nunamaker, Chen and Purdin (1991, pp. 95-96) state: "where multiple methodologies are applicable, they appear to be complementary, providing valuable feedback to one another." This study uses qualitative information in order to prepare an explanatory theoretical paper to study the key functions in managing software project effort. The paper reviews literature to describe, analyse and understand the software project effort activities.

DISCUSSION & CONCLUSION

Software engineering literature (Pressman, 2005; Royce, 1998; Sommerville, 2001) focuses on project management and effort concepts, but the emphasis is on effort estimation and effort-related planning (e.g., scheduling) rather than on the total management of effort. Effort has not been seen as an independent area of management like risk or quality. Literature discusses risk management, quality management and configuration management individually but effort is covered as a part of software project management. In other words, research has particularly focused on improving effort estimation, and has concentrated on the software construction effort of the software projects.

Several effort and cost estimation models and methods have been proposed during the last decades. Despite the numerous applications available for estimation, effort is, however, frequently underestimated (Gruschke and Jørgensen, 2008; Kitchenham, et al., 2009). According to studies (Briand, et al., 1999; Shepperd, et al., 1996), the effort estimation

methods and tools have reached only low accuracy levels with MPE-values worse than 0.30, whereas the target value between the estimated and actual effort for software industry companies can be three times smaller, for instance. The effort estimation research and the different estima-tion techniques emphasize two factors in respect to the difficulty in producing accurate estimates: software size and the available data for estimations (Armour, 2002). Most formal effort and cost estimation models and methods require a determination on the size of the software to be produced. The size is generally determined for example in terms of *lines of code* (LOC), *function points* (FP), or *use case points* (UCP). The prediction of the size of the final software product is, however, challenging and complex. Another factor which reflects to inaccurate effort estimates is the shortage of knowledge on the software to be produced since these estimates are made especially in the early stages of the software development process (Armour, 2002; Boehm, 1981).

Although several formal approaches have been proposed for effort estimation they are seldom employed in practice. As mentioned in literature, the applied effort estimation approaches are in most cases informal, and based on analogy and expert analysis. Jorgensen daunts that it might not be possible to develop effort estimation models that replace expert judgment, and therefore the best effort estimation improvement strategy may be to improve the judgment-based effort estimates (Jorgensen, 2005).

An overlooked, possible factor explaining poor effort estimates is the light consideration on different activities involved with the project. Therefore, the likelihood of unintentionally leaving out significant activities affecting the estimation is significant. Also, the focus in effort estimation research and estimation methods has been on software construction and project management (MacDonell and Shepperd, 2003), whilst presenting the other activities related to the project merely as a fixed proportion from the software construction effort, and as effort overhead. These activities, which are not directly related to software construction or project management, include various management and support activities, each carried out by several members of the project.

Experiences from the software industry imply that in effort estimations where expert judgment technique has been applied the estimators consider exclusively those activities that occur for them in a software project and focus on the effort of actual software construction. Indeed, research (e.g., (Jørgensen, 2004b, 2005; Jørgensen and Sjøberg, 2004)) confirms this observation. Also, there is a correlation between years of experience and the variety of different project activities considered for the estimate (Jørgensen and Sjøberg, 2004) more experienced experts consider a larger variety of project activities. Different experts estimate different areas, e.g., programmers estimate the amount of programming effort.

A reason for the effort estimate inaccuracy can be the inadequacy of previous projects' effort data collection when effort is both reported badly and collected without analysing it properly afterward. The collected effort data is used for both improving team performance in upcoming project phases and project and in calibrating the weights that adjust the factors and drivers which are used for the effort estimate derivation in the organization in question.

The proposed processes for post-mortem analysis (Collier, et al., 1996) describe post-mortem of the whole project, and provide general guidelines without a detailed method to analyse effort. Furthermore, the proposed analysis processes require rather large resources (both time and personnel) which, in practice, are quite limited in the software industry.

Thus from the effort-related challenges described above in the sub-chapters and theoretical background section, it can be concluded that *Estimators consider mostly core construction activities, relevant in their work and when estimating and Effort data quality is bad, i.e., it is not reliable, Effort and cost estimates are inaccurate. Also the estimator cannot judge the accurate result.*

REFERENCES

[1]. Agarwal, P., Kumar, M., Yogesh, Mallick, S., Bharadwaj, P.M. & Anantwar, D., 2001. Estimating software projects. *ACM JIGJOFT Software Engineering Notes*, 26(4), pp. 60-67.
[2]. Albrecht, A.J. & Gaffney, J.E., 1983. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Trans- actions on software Engineering*, 9(6), pp. 639-648.
[3]. Armour, P., 2002. Ten Unmyths of Project Estimation. *Communications of the ACM*, 45(11), pp. 15-18.
[4]. Awazu, Y., Desouza, K.C. & Evaristo, J.P., 2004. Stopping Punaway Information Technology Projects. *Business Horisons*, 47(1), pp. 73-80.
[5]. Blackburn, J.D., Scudder, G.D. & Van Wassenhove, L.N., 1996. Improving Speed and Productivity of Software Development: A Global Survey of Software Developers. *IEEE Transactions on Software Engineering*, 22(12), pp. 875-885.
[6]. Boehm, B., Clark, B., Horowitz, E., Westland, E., Madachy, P. & Selby, P., 1995. Cost Models for Future Life Cycle Processes: COCOMO 2. *Annals of software Engineering*, 1, pp. 57-94.
[7]. Boehm, B., Horowitz, E., Madachy, P., Peifer, D., Clark, B., Steece, B., Brown, A., Chulani, S. & Abts, C., 2000. *software Cost Estimation with COCOMO II*. Upper Saddle Piver, NJ: Prentice-Hall.

[8]. Boehm, B.W., 1981. *software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall.
[9]. Briand L.C., El Emam, K., Surmann, D., Wieczorek, I. & Maxwell, K.D., 1999. An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques. In: *21st International Conference on Software Engineering (ICJE'99)*, Los Angeles, California, USA, 16-22 May 1999. ACM, pp. 313-322.
[10]. Briand, L.C., Langley, T. & Wieczorek, I., 2000. A replicated Assessment and Comparison of Common Software Cost Modeling Techniques. In: *22nd Interna- tional Conference on Software Engineering (ICJE'00)*. Limerick, Ireland, 4-11 June 2000. ACM Press, pp. 377-386.
[11]. Brooks, F.P. Jr., 1975. *The Mythical Man-Month. Essays on software Engineering*. Peading, MA: Addison-Wesley Publishing Company.
[12]. Chatters, B., Henderson, P. & Postron, C., 1999. An Experiment to Improve Cost Estimation and Project Tracking for Software and Systems Integration Projects. In: *25th Euromicro Conference (E5ROMICRO '99)*. Milan, Italy, 8-10 September 1999. IEEE Computer Society, pp. 2177- 2184.
[13]. Chen, Z., Menzies, T., Port, D. & Boehm, B., 2005. Finding the Right Data for Software Cost Modeling. *IEEE Software*, 22(6), pp. 38-46.
[14]. Cohn, M., 2008. *Agile Estimating and planning*. Upper Saddle Piver, NJ: Prentice Hall.
[15]. Collier, B., DeMarco, T. & Fearey, P., 1996. A Defined Process for Project Post-Mortem Peview. *IEEE Software*, 13(4), pp. 65-72.
[16]. Conte, S.D., Dunsmore, H.E. & Shen, V.Y., 1986. *Software Engineering Metrics and Models*. Menlo Park, CA: Benjamin/Cummings Publishing Company.
[17]. Fairley, P.E., 1992. Pecent Advances in Software Estimation Techniques. In: *14th International Conference on Software Engineering*. Melbourne, Australia, 11-15 May 1992. ACM Press, pp. 382-391.
[18]. Fenton, N.E. & Pfleeger, S.L., 1997. *Software Metrics: A Rigorous & practical Approach*. 2nd ed. Boston, MA: PWS Publishing Company.