An Exploratory Survey of Phase-wise Project Cost Estimation Techniques

Peraphon Sophatsathit

Advanced Virtual and Intelligent Computing (AVIC) Center
Department of Mathematics and Computer Science
Faculty of Science, Chulalongkorn University, Bangkok, Thailand
E-mail: <speraphon@gmail.com>

Abstract

This article explores a number of existing project cost estimation techniques to investigate how the estimation can be done in a more accurate and effective manner. The survey looks into various estimation models that utilize many theoretical techniques such as statistics, fuzzy logic, case-based reasoning, analogies, and neural networks. As the essence of conventional estimation inaccuracy lies in life cycle cost drivers that are unsuitable to be applied across the project life cycle, this study introduces a phase-wise estimation technique that posits some overhead and latency costs. Performance evaluation methods of the underlying phase-wise principle are also presented. Contributions of this phase-wise approach will improve the estimation accuracy owing to less latency cost and increase the project visibility which in turn helps the project managers better scrutinize and administer project activities.

Keywords*: Phase-wise effort estimation, fine-grained estimation, cost driver, overhead cost, latency cost.*

1. Introduction

Software project cost estimation has plagued the software industry since the early years of software engineering. From the call for bidding to closing of project, the success and failure are attributed to how well the project is managed to keep everything in perspective, in particular, not exceeding the budget. Many projects ended with bad notes. The culprit lies in the root cause of cost estimation which is primarily derived from effort estimation. In a typical project management process, the fact that the project life cycle (LC) spans over a long period of time makes it hard to accurately estimate the cost. Myriad of unanticipated risks and uncertainties, not to mention the essence and accidents of software engineering (Brooks 1987) and the metrics of software complexity (Yu and Zhou 2010), often cause under- or over-estimation that leads to project loss and termination. A couple of compelling questions that project managers often ask themselves are what to measure and how to accurately measure it. Experience tells us that both project effort and metrics are means to the solutions of the above questions. Unfortunately, the answer to the first question cannot be precisely and directly determined in advance so that planning of the project can be carried out systematically. In the meantime, the attempt to answer the second question calls for some forms of cost estimation modeling in order to arrive at an educated guess that furnishes necessary and pertinent information to plan the project. The conventional state-of-the-practice usually employs classical or established project LC models for effort estimation such as waterfall model, spiral model, SLIM model/Putnam model (Putnam 1978), COnstructive COst MOdel (COCOMO 81 (Boehm 1981) and COCOMO II (Boehm *et al*. 2000), Walston-Felix model (Walston and Felix 1977), Bailey-Basili model (Bailey and Basili 1981), Albrecht-Gaffney model (Albrecht and Gaffney 1983), Kemerer empirical model (Kemerer 1987), and Matson, Barrett and Mellichamp model (Matson *et al*. 1994). These models culminate in practical cost estimation based on well-defined software metrics, namely, lines of code (LOC), function point (FP) (Albrecht 1979), use case point (UCP) (Karner 1993), application point (AP) (Boehm *et al*. 2000), delivered source instructions

(DSI), unadjusted use case point (UUCP), etc. All of these approaches use the project lifecycle in the estimation process. The results unfortunately yield a number of estimation errors such as phase overhead and phase-transition overhead. These cost factors hereafter will be referred to as latency overheads. The above undertakings are still far from gaining accurate estimation. The measurement accuracy is still a subject to validation that involves a wide array of techniques. Research approaches are underway to attain accurate effort estimation, for instance, statistical inferences, parametric modeling, knowledge-based modeling, case-based reasoning, fuzzy logic, neural networks, and analogies (Shepperd and Schofield 1997; Li *et al*. 2007). The question is how well the methods proposed by these models perform as far as the measurement error from the data sets and the comparative performance statistics with other techniques are concerned.

This paper surveys alternate approaches on project cost estimation based on current and emerging research endeavors. The survey encompasses various research attempts and emerging viewpoints that propose a breakdown of the project LC in order to perform fine-grained estimation at phase level. As such, the total project effort can be determined in a more accurate manner. The pros and cons of these approaches and viewpoints will be applicable to actual project management and remain to be validated during project implementation.

This paper is organized as follows. Section 2 recounts some principal building blocks that are applied in many studies. Section 3 describes phase-wise estimation approaches that apply to the conventional project LC. Suggestions, prospects and future directions, along with some final thoughts, are given in the Conclusion.

2. Literature Review

There have been a number of research endeavors to investigate project effort estimation (Nasir 2006). A number of models, techniques, tools, and metrics are used by the principal investigators to improve the measurement accuracy of the estimation based on the most predominant cost driver, that is, project size. The introduction of machine learning has brought the researches in the area to a new level. Integrated techniques, multi-methods, and extensive measurement schemes are being applied. Table 1 depicts a short chronological list of such research works.

Table 1. A chronological review of effort estimation research works.

Author(s) (Year)	Primary focus	Approaches	Evaluation
Kocaguneli *et al*. (2012)	Ensemble	Multimethods/solo method	MAR, MMRE, MdMRE, MMER, PRED(25), MBRE, MIBRE
Dejaeger *et al*. (2012)	Data mining	OLS, OLS+log, OLS+BC, Robust regression, Ridge regression, Least median squares, MARS, CART, Model tree, MLP NN, RBFN, CBR, LS+SVM	MdMRE, PRED(25), Spearman's rank correlation
Port and Korte (2008)	Model evaluation	Models	MMRE, PRED
Yang *et al*. (2008)	Phase distribution	Models	KSLOC
Li *et al*. (2007)	Analogy	SIM	MMRE, PRED(25)
Matson *et al*. (1994)	Function point	Models	MRE
Kemerer (1987)	Empirical model	KSLOC	MRE
Albrecht and Gaffney (1983)	Function point, source lines of code	FP, KSLOC	MRE
Boehm (1981) / Boehm *et al*. (2000)	COCOMO	KDSI, KSLOC	LOC
Albrecht (1979)	Function point	FP	UFP
Putnam (1978)	SLIM	Norden/Rayleigh	Regression

As software becomes intertwined in every facet of our lives, its application has been internationally accepted and used. The needs for standardizing its deliverables and development process are key factors to software products. Typical acclaimed standards and working groups (WGs) are:

- 16326 WG - Information Technology - Software Engineering - Software Project Management Working Group, Institute of Electrical and Electronics Engineers (IEEE);
- European Research Consortium for Informatics and Mathematics (ERCIM) Working Group Software Evolution;
- Project Management Institute (PMI) and Project Management Body of Knowledge (PMBOK®);
- International Organization for Standardization (ISO)/ International Electrotechnical Commission (IEC) 14143 [six parts] Information Technology - Software Measurement - Functional Size Measurement;
- ISO/IEC 19761:2011 Software Engineering - Common Software Measurement International Consortium (COSMIC): A Functional Size Measurement Method;
- ISO/IEC 20926:2009 Software and Systems Engineering - Software Measurement - International Function Point Users Group (IFPUG) Functional Size Measurement Method;
- ISO/IEC 24570:2005 Software Engineering - Netherlands Software Metrics users Association (NESMA) Functional Size Measurement Method Version 2.1 - Definitions and Counting Guidelines for the Application of Function Point Analysis; and
- ISO/IEC 20968:2002 Software engineering - Mark-II Function Point Analysis - Counting Practices Manual.

Extensive as they are, the above attempts are based on the wide span of the project LC that encompasses various unscrupulous activities of the team. Some emerging viewpoints on effort estimation have gradually been investigated to unveil latency overheads. The empirical findings usually are inapplicable for accurate estimation purposes and the efforts are expended to focus on specific areas of application. User groups such as IFPUG and ISBSG (International Software Benchmarking Standards Group) are examples of domain specific estimation. The project managers must decide what data are to be collected to suit the applicable domain. For example, if the project managers decide to adopt the Constructive Cost Model (COCOMO), they have to predict the equivalent LOC for all upstream phases, i.e., requirements, planning, and design.

Yang *et al*. (2008) proposed a phase distribution of the software development effort showing where the workload is concentrated. MacDonell and Shepperd (2003) proposed the use of a completed project to predict the effort needed for subsequent activities. Jodpimai *et al*. (2010) explored the interrelationship among different dimensions of projects. The inherent characteristics of some activities are too subjective to be measured either by overall or discrete elements, thereby rendering inaccurate estimation. Thus, breaking up in phases may involve activities that cannot be directly measured while traditional LC estimation averages out such deficits in terms of an overall figure. On the other hand, the measureable ones often yield different project data based on different models, metrics, and cost drivers. A proposed novel framework on phase-wise estimation breakdown in the conventional LC model will be elucidated in the sections that follow.

3. Phase-wise Approach in the Conventional Lifecycle Model

There are many factors that must be taken into account when selecting an estimation technique suitable for a given project. The most important one is the designated project LC model whose applicability depends primarily on several project characteristics and dependencies. The characteristics encompass attributes such as complexity, development paradigm (object-oriented, functional, agile) and mode (organic, semi-detached, embedded), while the dependencies include size, technology and

metrics. In this survey, a classical project cost estimation based on COCOMO II is employed to demonstrate the combination of cost factors involved in the estimation. Table A-1 in the Appendix shows several necessary parameters for estimation. However, the number of parameters has been a formidable obstacle to accurate effort estimation after project inception. Several recent research findings (Kocaguneli *et al*. 2012; Dejaeger *et al*. 2012) advocate an ensemble of multiple estimation methods taking joint attributes into account. Kocaguneli *et al*. (2012) contended that there exist best combinations of methods for effort estimation. Nonetheless, the findings encompassed too many features to estimate, thereby the prediction accuracy had to be evaluated with more stringent criteria with the help of error evaluators such as mean magnitude of relative error (MMRE), percentage of relative error deviation (PRED) (Port and Korte 2008; Foss *et al*. 2003), etc. As mentioned earlier, most approaches deal with overall numbers so that the underlying estimation techniques cannot be generalized from empirical findings to production. The fact that different projects possess their own process variations renders different project effort estimations. This affects the accuracy of the conventional LC estimation model. The main deviation of the phase-wise approach is in phase dependent cost drivers that differentiate it from the conventional estimation. In the conventional approach, same constituent cost drivers are used to estimate the entire LC effort and the corresponding project cost. In contrast, the phase-wise approach utilizes selective cost drivers depending on phase requirements, nature of work, and deliverables. For instance, the project size is unarguably the principal estimator that is used throughout the entire project LC. If the project is broken down into a fine grain of measures on a phase basis, the project size may no longer be the appropriate cost estimator of the feasibility study and requirement analysis phases. This technique is essential and attributive in the establishment of the proposed phase-wise estimation. The following breakdown as shown in Fig. 1 elucidates a typical workflow of the phase-wise estimation.

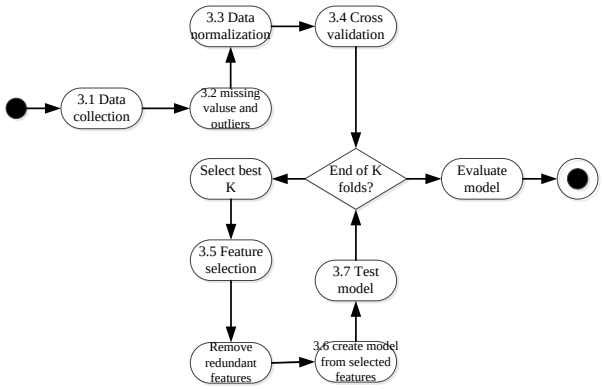

Fig. 1. Phase-wise effort estimation flow.

3.1 Data Collection

The proposed novel phase-wise approach will utilize the conventional waterfall model where each phase is well defined and known by all software developers. Due to the unprecedented phase-wise effort estimation approach, industrial data, especially existing standard benchmarking archives such as COCOMO81, NASA93, Desharnais, USP05, MAXWELL, etc., were available (PROMISE 2014) in overall LC figures. This survey will explore how the proposed novel approach can be put to production use.

The process begins by collecting data of each phase. The uppercase phases, i.e., requirements, specifications and design, will employ functional size measurement (Gencel and Demirors 2008) and the like as techniques for effort estimation.

The lowercase phases, on the other hand, can incorporate LOC metrics as additional details are available for fine-grained estimation. This sizing scheme may, at the first glance, appear to be a mixture of FP and LOC. Fortunately, the former metric can be converted to the latter via backfiring (Jones 1995).

Table 2 depicts a typical series of activities during the course of project execution. The information can then be used to estimate the phase-wise effort. The overall project figure is obtained from the sum of constituent phases. As such, any "latency overhead" will be uncovered on an individual phase basis to yield fine-grained effort estimation.

Table 2. Project phase-wise activities and metrics.

Activity	Input	Output	Estimation metrics
Software Requirements Specification (SRS)	No. of team members	Functionality	FP
Design	No. of requirements/member	Functional model	FP
Coding	Design document	Code	LOC
Testing	No. of reworks	No. of errors/module	LOC
Implementation	No. of transactions from production code	DSI	LOC

3.2 Missing Value and Outlier

This step is designed to serve as an early data correction exercised immediately after data collection. Tsunoda *et al*. (2011) exploited such early warnings of two problems, namely, missing value and outlier detection. The former is an inherent phenomenon in project management where numerous small activities are accounted for such as color change of status bar, alternate display of normal time to military time, or renaming of a counter variable to make it more meaningful, etc. The efforts spent on corrections are insignificant to be recorded but appear on requests for change or maintenance reports. The latter, on the contrary, result from exceptional situations where unusually high or low values are recorded, for example heavy fire damage in the data center that caused considerable recovery effort and project delay, or the arrival of a new and highly efficient integrated development environment (IDE) tool that helped speed up the development time in considerably less effort, yielding lower actual effort spent than already estimated.

One of the most popular and effective techniques to handle missing values is the *k*-nearest neighbor (*k*-NN) imputation (Keller *et al*. 1985). It uses *k* data points from the closest proximity of the missing value position to interpolate the most likely value. This imputation might incur some errors if the actual values were not missing. The use of same feature value from other projects permits cross validation that fills in the estimation of the missing value to yield more accurate results. This technique will be explained in subsequent steps.

The measuring error is computed as the Euclidean distance between the project having missing values and the ones without. A small value of the average of *N* measurements signifies accurate prediction of missing values.

Outlier detection can be typically handled by examining the kurtosis and skewness of the data distribution. A normality test is set up to be the null hypothesis based on *z*-score to determine if there are possibilities that said null hypothesis is accepted, i.e., *p*-value < 0.001. On the contrary, if the null-hypothesis is rejected, the highest value is treated as the outlier and is discarded. This process is repeated until all outliers are removed.

3.3 Data Normalization

The above two steps merely acquire input data from various sources which could be of different ranges and magnitudes. A standard technique to linearly scale data of different ranges to the same scale, yet still preserve the relationship with the original value, is carried out by Eq. (1) as follows:

$$\hat{x} = \frac{(x - x_{\min})}{(x_{\max} - x_{\min})} * (\hat{x}_{\max} - \hat{x}_{\min}) + \hat{x}_{\min}, \qquad (1)$$

where $\hat{x}$ denotes the reference range, while x denotes the individual range.

3.4 Cross-Validation

In a typical experiment, the data so collected are divided into two non-overlapping sets, that is, a training set and a test set. The former is used in various parameter adjustments during the model creation process, whereas the latter is held out for the model validation process. There are several techniques available for cross validation purposes. Two popular ones to be introduced are the *N*-fold cross validation and leave-one-

out cross validation techniques. In *N*-fold cross validation, training data are randomly partitioned into *N* subgroups. One group is used as the test set and the rest are training sets. This process is repeated *N* times. The leave-one-out approach exhaustively uses one data point at a time to test data and leave the rest of the data to be the trained set. This approach is very expensive and often used in a small number of data points as opposed to the *N*-fold technique that requires more data points to support the random rotation process.

3.5 Feature Selection

This is perhaps the most important activity of project cost estimation. Since all participating cost factors in the estimation process are features being selected from the cost model, many existing estimation techniques utilize several cost factors as estimation parameters. For example, the COCOMO model (Boehm *et al*. 2000) uses 17 cost drivers in the estimation process. Jodpimai *et al*. (2010) found that only a handful of cost drivers were effective factors that could derive an estimation as accurate as the comparative models without employing the full-fledged parameter set. Moreover, fewer cost drivers translated into faster computation time. The findings revealed some interesting consequences, e.g., certain features are vital cost drivers that could yield accurate estimation.

The process of acquiring good selected features is straightforward. The first step is to eliminate independent features that do not contribute or affect the project effort estimation. The next step is to reduce all redundant features that are less relevant to the project effort estimation. This is done by means of Pearson's correlation. Features that result in a low value will be less relevant and thus eliminated. Finally, only those features that are related to effort estimation in the form of a mathematical function will be retained (Jodpimai *et al*. 2010).

There are a number of models that can be used in the feature selection process, ranging from conventional COCOMO, rational unified process (RUP), statistical, and neural network models. The basis for effort estimation must rest on the proper use of these selected features in the estimation process, thereby accurate estimation results can be obtained. Table 3 depicts typical phase-wise features participating in cost estimation.

Table 3. Common phase-wise costing features.

Group	Feature	Phase	Attribute/activity
Uppercase	Software complexity	System analysis	- Overview analysis
		System design	- Documentation
	Analyst capability	Architectural design	- Design overview
		Detailed design	- I/O design - Data/class design
		Coding	- Program/module structure
Lowercase	Execution time	Testing	- Programming - Unit and integration testing
	Main storage	Production	- Installation - Use/implement
	Programmer capability	Coding	- Syntactic and logical details

3.6 Performance Evaluation

Some popular performance evaluation methods and their corresponding effort estimation metrics collected in this survey are summarized in Table 4. Each method has its own applicability to measure the relationship between actual and predicted estimation results based on the set of selected features. This in turn yields accuracy of the estimation model being employed.

Table 4. Common methods/metrics for performance evaluation.

Metric	Name	Reference	Remark
MRE	Magnitude of Relative Error	Tsunoda *et al*. (2011)	$\lvert y'_i - y_i \rvert / y_i$
BRE	Balanced Relative Error	Kitchenham *et al*. (2001)	$\lvert y'_i - y_i \rvert / \min(y'_i, y_i)$
MER	Magnitude of Error Relative	Foss *et al*. (2003)	$\lvert y'_i - y_i \rvert / y'_i$
MMRE	Mean Magnitude of Relative Error	Tsunoda *et al*. (2011)	$\frac{1}{n}\sum_{i=1}^{n} MRE$
PRED(l)	Prediction at Level l	Tsunoda *et al*. (2011)	$\frac{1}{n}\sum_{i=1}^{n} \begin{cases} 1 & if\ MRE_i \le l \\ 0 & otherwise \end{cases}$
MdMRE	Median Magnitude of Relative Error	Friedman (1937)	median (*MRE*)
Pearson's correlation	Relation between two sets of data (estimated and actual)	Abran and Robillard (1996)	$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$
Friedman test	Non-parametric hypothesis test as an alternative to one-way ANOVA	Friedman (1937)	$\bar{r} = \frac{1}{nk}\sum_{i=1}^{n}\sum_{j=1}^{k} r_{ij}$
Wilcoxon matched pairs signed-rank test	Non-parametric as an alternative to parametric paired simple *t*-test	Rédei (2008)	$\lvert x_{2,i} - x_{1,i} \rvert$, sgn($x_{2,i} - x_{1,i}$)
Kruskal-Wallis analysis of variance (ANOVA) test	Non-parametric as alternative one-way ANOVA > 3 samples	Vries and Chandon (2007)	$K=(N-1)\frac{\sum_{i=1}^{g} n_i (r_i - r)^2}{\sum_{i=1}^{g}\sum_{j=1}^{n_i} (r_{ij} - r)^2}$

Model evaluation is usually carried out by comparing the difference between predicted (estimated) effort y'_i and actual effort y_i. The effort is made on a phase basis using related factors. For example, factors used in requirements and specification effort estimation involve FP to deal with both functional and non-functional requirements. As project requirements become materialized, size estimation via LOC can be carried out more accurately. Some metrics are criticized for penalizing over-estimation (such as MRE), while others are just the opposite (such as BRE and MER). Two commonly used metrics are MMRE and PRED (0.25) since they are independent of units of measure and easy to use. However, they are found to give inconsistent results depending on the properties of the distribution of y'_i/y_i. Occasionally, MdMRE is used to solve the outlier problem as MMRE cannot properly handle it. At any rate, these two metrics will be adopted.

Since this survey also introduces a new approach using phase-wise estimation to predict fine-grained individual phase level cost as opposed to the conventional overall LC estimation, the metrics used are confined to the same ones in order to keep the same working parameters so that the results so obtained will be comparable with those of the conventional statistical techniques (Kitchenham *et al*. 2001).

4. Future Directions

This survey sets out to explore phase-wise project cost estimation, aiming at more accurate results than using the traditional LC approach. The survey unveils some noteworthy results that have potential benefits to software project management.

Modern software products are short-lived. Their development process cannot fit into the traditional LC style, let alone analysis and management. The proposed phase-wise approach could open a new exploitation of well-entrenched and practical techniques that have already been put to use. This in turn becomes an adoption of proven techniques such as program slicing to new software development techniques or management approaches. The phase-wise analysis can serve

as a scrutinized investigation provided that one can find appropriate metrics and analysis technique to handle it.

The essence of this phase-wise estimation survey can be summarized as follows:

1) The phase-wise measurement is an inherent rather than accidental characteristic of project management. The nature of the software development process irrespective of the underlying model lends itself to collecting data which are readily available. What has not been done is the breakdown of the activity structure. In addition, the difficulty of data collection is seen as disruptive and infeasible as many tasks are operationally intertwined. This makes it hard to succinctly separate.
2) The fact that a "phase" has been ingrained in software engineering and has become a stigma of software project management makes it unfit to new software project development paradigms such as agile and aspect-oriented approaches. A closer look into this survey reveals that this *phase* can be generalized to fit the underlying development process. For example, for an input screen storyboard of an agile development setting, the deliverables can be viewed as one UI work product of the input-output phase which is appropriately measured as phase-wise results.
3) Existing available tools, techniques, and metrics can be tailored to suit the phase level without having to reinvent the wheel. For example, project management tools can easily be set to monitor the phase-wise measurement.
4) New development paradigms, technologies, and management techniques are required for the success of phase-wise measurement. As such, training to work with new procedures and assessments is inevitable. Proper planning must be carried out to set up the necessary programs for all personnel involved. Moreover, the activity measure will be broken down to fine man-hour levels to handle a smaller scale of required items.

The current work in progress using a collection of industrial projects looks quite promising as the number of cost factors is reduced considerably without sacrificing estimation accuracy.

5. Conclusion

This survey examines a number of existing software project cost estimation techniques and metrics from the project life cycle standpoint. All research estimation works are evaluated using well-established measurement statistics. At any rate, the inherent estimation error still persists. A novel phase-wise approach is proposed as a preliminary investigation to explore the intricacy of project effort measurement so as to arrive at more accurate estimation. A number of benefits can be drawn from such an elaborative approach:

1) One of the most important software project management aspects is project visibility. The project managers will be able to monitor any activities or actions that have gone awry in each phase and transition between phases rather than prolonging them until a catastrophic failure takes place.
2) More appropriate performance measures and metrics can be applied in each designated phase as opposed to using the traditional "one size fits all" metric on the entire LC.
3) Fewer cost drivers are used. This means faster and more accurate estimation than the traditional LC approach as the cumulative errors grow from being injected by too many cost drivers.
4) The phase-wise activity breakdown offers reduction or elimination of latent and transition costs with better cost handling. The project managers will be able to see the hidden problems, effort duplication, and overhead incurred within and between phases.

This precursory survey furnishes a preliminary overview of fine-grained measurements in project cost estimation. The project managers will have less of a burden on cost estimation and more time to monitor relevant project activities.

6. References

Abran, A.; and Robillard, P.N. 1996. Function points analysis: an empirical study of its measurement processes. IEEE Transactions on Software Engineering 22(12): 895-910.

Albrecht, A.J. 1979. Measuring application development productivity. Proc. Joint SHARE, GUIDE, and IBM Application Development Symposium, Monterey, CA, USA, 14-17 October 1979. Pp. 83-92. IBM Press, Indianapolis, IN, USA.

Albrecht, A.J.; and Gaffney, J.E., Jr. 1983. Software function, source lines of code, and development effort prediction: a software science validation. IEEE Transactions on Software Engineering SE-9(6): 639-48.

Bailey, J.W.; and Basili, V.R. 1981. A meta-model for software development resource expenditures. Proc. 5th Int. Conf. on Software Engineering (ICSE), San Diego, CA, USA, 9-12 March 1981. Pp. 107-16. IEEE Press, Piscataway, NJ, USA.

Boehm, B.W. 1981. Software Engineering Economics. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Boehm, B.W.; Abts, C.; Brown, A.W.; Chulani, S.; Clark, B.K.; Horowitz, E.; Madachy, R.; Reifer, D.J.; and Steece, B. 2000. Software Cost Estimation with COCOMO II. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Brooks, F.P., Jr. 1987. No silver bullet - essence and accidents of software engineering. Computer 20(4): 10-9.

COCOMO II. 2000. Constructive Cost Models (COCOMO) II - Model Definition Manual. Version 2.1, 1995-2000. Center for Software Engineering, School of Engineering, University of Southern California (USC), Los Angeles, CA, USA. Available: <http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_modelman2000.0.pdf>.

Dejaeger, K.; Verbeke, W.; Martens, D.; and Baesens, B. 2012. Data mining techniques for software effort estimation: a comparative study. IEEE Transactions on Software Engineering 38(2): 375-97.

Foss, T.; Stensrud, E.; Kitchenham, B.; and Myrtveit, I. 2003. A simulation study of the model evaluation criterion MMRE. IEEE Transactions on Software Engineering 29(11): 985-95.

Friedman, M. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association 32(200): 675-701.

Gencel, C.; and Demirors, O. 2008. Functional size measurement revisited. ACM Transactions on Software Engineering and Methodology 17(3): Article 15, 36 pp.

Jodpimai, P.; Sophatsathit, P.; and Lursinsap, C. 2010. Estimating software effort with minimum features using neural functional approximation. Proc. Int. Conf. on Computational Science and Its Applications (ICCSA), Fukuoka, Japan, 23-26 March 2010. Pp. 266-73. IEEE Computer Society, Los Alamitos, CA, USA.

Jones, C. 1995. Backfiring: converting lines of code to function points. Computer 28(11): 87-8.

Karner, G. 1993. Resource estimation for objectory projects. Objective Systems SF AB, Kista, Stockholm Municipality, Sweden, 17 September 1993. Available: <http://www.larmel.net/upload/Karner - Resource Estimation for Objectory Projects_56143.pdf>.

Keller, J.M.; Gray, M.R.; and Givens, J.A. 1985. A fuzzy *K*-nearest neighbor algorithm. IEEE Transactions on Systems, Man and Cybernetics SMC-15(4): 580-5.

Kemerer, C.F. 1987. An empirical validation of software cost estimation models. Communications of the ACM 30(5): 416-29.

Kitchenham, B.A.; Pickard, L.M.; MacDonell, S.G.; and Shepperd, M.J. 2001. What accuracy statistics really measure [software estimation]. Proc. IEE Proceedings - Software 148(3): 81-5.

Kocaguneli, E.; Menzies, T.; and Keung, J.W. 2012. On the value of ensemble effort estimation. IEEE Transactions on Software Engineering 38(6): 1,403-16.

Li, J.; Ruhe, G.; Al-Emran, A.; and Richter, M.M. 2007. A flexible method for software effort estimation by analogy. Empirical Software Engineering 12(1): 65-106.

MacDonell, S.G.; and Shepperd, M.J. 2003. Using prior-phase effort records for re-estimation during software projects. Proc. 9th Int. Software Metrics Symposium. (METRICS), Sydney, Australia, 3-5 September 2003. Pp. 73-86. IEEE Computer Society, Los Alamitos, CA, USA.

Matson, J.E.; Barrett, B.E.; and Mellichamp, J.M., 1994. Software development cost estimation using function points. IEEE Transactions on Software Engineering 20(4): 275-87.

Nasir, M. A survey of software estimation techniques and project planning practices. Proc. 7th ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD), Las Vegas, NV, USA, 19-20 June 2006. Pp. 305-10. IEEE Computer Society, Los Alamitos, CA, USA.

Port, D.; and Korte, M. 2008. Comparative studies of the model evaluation criterions MMRE and PRED in software cost estimation research. Proc. 2nd ACM-IEEE Int. Symposium on Empirical Software Engineering and Measurement (ESEM), Kaiserslautern, Germany, 9-10 October 2008. Pp. 51-60. Association for Computing Machinery (ACM), New York, NY, USA.

PROMISE. 2014. Public Datasets. Predictor Models in Software Engineering (PROMISE) Software Engineering Repository, School of Information Technology and Engineering, University of Ottawa, Ottawa, Ontario, Canada. Available: <http://promise.site.uottawa.ca/SERepository/datasets-page.html >.

Putnam, L.H. 1978. A general empirical solution to the macro software sizing and estimating problem. IEEE Transactions on Software Engineering SE-4(4): 345-61.

Rédei, G.P. 2008. Encyclopedia of Genetics, Genomics, Proteomics and Informatics. Volume 1: A-L. Volume 2: M-Z. 3rd ed. Springer, Dordrecht, The Netherlands.

Shepperd, M.; and Schofield, C. 1997. Estimating software project effort using analogies. IEEE Transactions on Software Engineering 23(11): 736-43.

Tsunoda, M.; Kakimoto, T.; Monden, A.; and Matsumoto, K.-I. 2011. An empirical evaluation of outlier deletion methods for analogy-based cost estimation. Proc. 7th Int. Conf. on Predictive Models in Software Engineering (PROMISE). Banff, Canada, 20-21 September 2011. Article 17, 10 pp. Association for Computing Machinery (ACM), New York, NY, USA.

de Vries, D.K.; and Chandon, Y. 2007. On the false-positive rate of statistical equipment comparisons based on the Kruskal-Wallis *H* statistic. IEEE Transactions on Semiconductor Manufacturing 20(3): 286-92.

Walston, C.E.; and Felix, C.P. 1977. A method of programming measurement and estimation. IBM Systems Journal 16(1): 54-73.

Yang, Y.; He, M.; Li, M.; Wang, Q.; and Boehm, B. 2008. Phase distribution of software development effort. Proc. 2nd ACM-IEEE Int. Symposium on Empirical Software Engineering and Measurement (ESEM). Kaiserslautern, Germany, 9-10 October 2008. Pp. 61-9. Association for Computing Machinery (ACM), New York, NY, USA.

Yu, S.; and Zhou, S. 2010. A survey on metric of software complexity. Proc. 2nd IEEE International Conference on Information Management and Engineering (ICIME), Chengdu, China, 16-18 April 2010. Vol. 2, pp. 352-6. IEEE Press, Piscataway, NJ, USA. Institute of Electrical and Electronics Engineers, Inc. Representative Office, Beijing, China.

Appendix

Table A-1. Cost factor/driver parameters for use in effort estimation.

Factor	Measurement	Level	Description	Value
Effort Adjustment Factor (EAF)	Product of cost drivers that do not use average value	a * (KSLOC)^b * EAF	SLOC/1000	*a*=2.94, *b*=0.91+(Σscale factors/100) [§]
Scale	PREC - precedentedness	Highest	Extremely familiar	0.00
		Very high	Very familiar	1.24
		High	Familiar	2.48
		Average	Somewhat familiar	3.72
		Low	Inexperienced but slightly understand	4.96
		Very low	No experience	6.20
	FLEX - development flexibility	Highest	Meet requirements	0.00
		Very high	Meet some requirements	1.01
		High	Consistent	2.03
		Average	Somewhat consistent	3.04
		Low	Slightly consistent	4.05
	RESL - architecture/risk resolution	Highest	Good planning	0.00
		Very high	N/A	1.41
		High	N/A	2.83
		Average	N/A	4.24
		Low	N/A	5.65
		Very low	No planning	7.07
	TEAM - team cohesion	Highest	Excellent communication	0.00
		Very high	Very good communication	1.10
		High	Good communication	2.19
		Average	Normal communication	3.29
		Low	Little communication	4.38
		Very low	Very little communication	5.48
	PMAT - process maturity	Highest	Capability Maturity Model Integration (CMMI) 5	0.00
		Very high	CMMI 4	1.56
		High	CMMI 3	3.12
		Average	CMMI 2	4.68
		Low	CMMI 1	6.24
Product	RELY - required reliability	Very high	Risky of life	1.26
		High	Big financial loss	1.10
		Average	Waste time to recover data	1.00
		Low	Delay in data recovery	0.92
		Very low	Some inconvenience	0.82
	DATA - database size (measured in bytes/SLOC)	Very high	Over 1,000	1.28
		High	$100 < x < 1000$	1.14
		Average	$10 < x < 100$	1.00
		Low	Less than 10	0.90
	CPLX - product complexity (control, computation, GUI)	Very high	Recursion, interrupt, 2D/3D, multimedia	1.34
		High	Nested loops, widgets, multimedia	1.17
		Average	Standard code and functions, simple widgets	1.00
		Low	Moderate code and functions	0.87
		Very low	Simple code and functions	0.73
	RUSE - required reusability	Highest	Reused with different application programs	1.24
		Very high	Reused with similar application programs	1.15

		High	Reused with same application but different project	1.07
		Average	Reused in different modules of the same project	1.00
		Low	Not reusable	0.95
	DOCU - documentation match to life cycle needs	Very high	Extremely important	1.23
		High	Very important	1.11
		Average	Appropriate	1.00
		Low	Incomplete	0.91
		Very low	Very few	0.81
Platform	TIME - execution time constraint	Very high	85%	1.29
		High	70%	1.11
		Average	<= 50%	1.00
	STOR - main storage constraint	Very high	85%	1.17
		High	70%	1.05
		Average	<= 50%	1.00
	PVOL - platform volatility	Very high	N/A	1.30
		High	Change every 2 months	1.15
		Average	Change every 6 months	1.00
		Low	Change every 12 months	0.87
Personnel	ACAP - analyst capability	Very high	90th percentile	0.71
		High	75th percentile	0.85
		Average	55th percentile	1.00
		Low	35th percentile	1.19
		Very low	15th percentile	1.42
	APEX - application experience	Very high	6 years	0.81
		High	3 years	0.88
		Average	1 year	1.00
		Low	6 months	1.10
		Very low	<= 2 months	1.22
	PCAP - programmer capability	Very high	90th percentile	0.76
		High	75th percentile	0.88
		Average	55th percentile	1.00
		Low	35th percentile	1.15
		Very low	15th percentile	1.34
	PLEX - platform experience	Very high	6 years	0.85
		High	3 years	0.91
		Average	1 year	1.00
		Low	6 months	1.09
		Very low	<= 2 months	1.19
	LTEX - language and tool experience	Very high	6 years	0.84
		High	3 years	0.91
		Average	1 year	1.00
		Low	6 months	1.09
		Very low	<= 2 months	1.20
	PCON - personnel continuity	Very high	3%	0.81
		High	6%	0.90
		Average	12%	1.00
		Low	24%	1.12
		Very low	48%	1.29

[§] For no scale factor, use b = 1.0997.
Source: COCOMO II (2000).