METHODS OF EFFORT ESTIMATION IN SOFTWARE ENGINEERING

Jovan Živadinović, Ph.D*
High school for business economy and entrepreneurship, Mitropolita Petra br.8, 11000 Beograd, Serbia
zjovan50@gmail.com

Zorica Medić
Ministry of Internal Affairs – Republic Serbia, Kneza Miloša br. 101, 11000 Beograd, Serbia

Dragan Maksimović
Ministry of Internal Affairs – Republic Serbia, Kneza Miloša br. 101, 11000 Beograd, Serbia

Aleksandar Damnjanović, M.Sc
High school for business economy and entrepreneurship, Mitropolita Petra br.8, 11000 Beograd, Serbia

Sladan Vujčić
High school for business economy and entrepreneurship, Mitropolita Petra br.8, 11000 Beograd, Serbia

ABSTRACT: The objective of this paper is to present the most relevant methods and models for effort estimation used by software engineers in the past four decades. Classification of the methods has been also suggested as well as brief description of the estimation methods presented.

Key words: estimation, effort, metrics

1. INTRODUCTION

In software engineering effort is used to denote measure of use of workforce and is defined as total time that takes members of a development team to perform a given task. It is usually expressed in units such as man-day, man-month, man-year. This value is important as it serves as basis for estimating other values relevant for software projects, like cost or total time required to produce a software product.

Reasons for effort estimation vary, some of the most frequent being:

- *Project approval.* There must be a decision on project launching on the part of an organization, preceded by effort estimation required for successful completion of the project.
- *Project management.* Project managers are responsible for planning and managing of the project. Both these activities require effort estimation as per respective phases in order for project to be completed.
- *Understanding by the development team members.* In order that development team could perform efficiently, it is necessary for its members to understand their individual roles as well as overall activities of the team as a whole.
- *Defining of project task,* that can be used for this purpose, is done by means of effort estimation.
- The accuracy of effort estimation is as current an issue for researchers today as it was 25 years ago when it was launched by Brooks (1975) in his work "The Mythical Man Month". Even today these estimations are mainly unreliable, with no proof of significant progress that has been made in their improvement, despite considerable funds and activities that have been invested to that purpose. Different authors classify effort estimation methods differently. Following classifications have been adopted for this purpose: Empirical parametric (algorithmic) estimation models; Empirical non-parametric estimation models; Expert estimation; Analogue estimation models; Downward estimation; Upward estimation".
It should be mentioned that these categories do not exclude one another. Furthermore, estimation methods exist which, in respect of some of their features, belong to different groups.

2. EMPIRICAL PARAMETRIC ESTIMATION MODELS

These models rely on the experience gained on previous software projects in the sense that they connect size and effort value by means of one of the explicit function forms, by applying regression analysis method. In doing so, most widely used are linear and exponential dependence. For example, product value measure can be the number of code lines in the programme or another value measure used to quantify some characteristics of the software product. Effort is usually expressed in values such as man-day or man-hour, sometimes man-year.

Good sides of these models are: objectivity, formalism, efficiency, and the fact that they have been based on the experience drawn from engineering practice. Its bad sides are: necessity for calibration before application in the concrete environment, subjectivity of input values, they have founding in the past instead of future.

Over the time, a great many empirical parametric estimation models have been developed resulting in attempts to determine the best among them which would be pronounced standard. However, these attempts yielded no result due to certain shortcomings characteristic for these models. Once the form of such a model is determined, one should have sufficient data to be able to establish the connection between the estimated value and mutually independent parameters in the model. Furthermore, it is not often quite clear whether all the parameters participating in the function are actually mutually independent. Also factors that have impact on estimated values are numerous and hard to precisely quantify.

When applying empirical parametric models attention should be paid to following: to make them as least complex as possible, to carefully collect relevant data and to gather as many practical examples as possible.

One of the best known empirical parametric models developed hitherto has following form:

\[
\text{Effort} = a \times \text{LOC}^b, \quad (1)
\]

where effort (man-months) represents work required to realize the system, while LOC is the number of code lines to be written. This model has been researched by Walston and Felix (1977), Bailey and Basili (1981) and (Boehm 1981), Boehm (2004) as the basis for COCOMO model. Walston and Felix have reached the value for \(b < 1\), while other researchers quote values \(b > 1\).

Kitchenham (1992), Kitchenham and Pearl Brereton (2010) in his research concludes that in the prevailing number of cases the value of \(b\) is close enough to 1 which justifies consideration of introducing linear dependence between the number of code lines and effort. In order to verify this theory, Banker, Chang and Kemerer (1994), used model form:

\[
\text{Effort} = a + b \times \text{LOC} + c \times \text{LOC}^2. \quad (2)
\]

In examining the data in respect of eleven software projects, they concluded that the coefficient \(c\) considerably differs from \(0\) in six cases. This proved the view that linear dependence of effort upon code lines can not be applied in a great number of of cases. Effort estimation models based on the number of code lines have one considerable shortcoming: the number of code lines is known only after the coding and testing, i.e. quite late in the lifecycle of software development. In order that these models could be used in the early phases, often an estimation of the number of code lines is made, followed by effort estimation. However, there also exist other metrics of software size which can be calculated in the earlier phases of the lifecycle than code lines.

The best known and most widely used metrics among them is function points metrics. A number of researchers, among them Albrecht and Gaffney (1983), (Kemerer 1987), Kemerer (1993) Matson, Barret and Mellichamp (1994), have examined models form:

\[
\text{Effort} = a + b \times \text{FP}, \quad (3)
\]

where Effort (man-months) is work required for the realization of the system and FP is the number of function points.

De Marco (1982) developed a model whereby effort required for software project implementation is estimated on the basis of data flow graph and object – relation graph, i.e. rather early in the software lifecycle. Metrics introduced by this researcher are called "function explosion" and "data explosion". The basic shortcoming of these metrics is that they are not obtained by means of direct counting, but by means of introducing rather complex weight coefficients, like with function points. What is more important, no published results exist in practice concerning the application of this estimation manner. Basili and Panililip-Yap (1985), (Basili et al., 1996), suggest a model that would be based on counting the number of pages of the system documentation

\[
\text{Effort} = a + b \times \text{NumberofpagesDocumentation}, \quad (4)
\]
where the pages of the written material describing the programme are counted, without taking into account the programme original code. Although the model, as the authors claim, showed rather good results concerning 23 research projects, it still failed to gain wider use for obvious reasons: the number of documentation pages is known even later than the number of code lines. In addition to this, the manner and volume of documented software projects do not vary not only from one organization to another but from one individual to another. This metric is affected even by factors such as font size used in documentation or page margin.

Brownlow (1994) researches effort estimation model that can be applied on object-oriented system analysis and design. It is based on the number of objects and services of the system. The form of the model researched is following:

$$\text{Effort} = a + b \text{Number of objects} + c \text{Number of services}$$  \hspace{1cm} (5)

The basic criticism that can be voiced concerning this model is its verification on a small number of projects. One of the arguments voiced by researchers denying the linear dependence of effort upon value is that the greater the system the greater the number of development team members as well as the greater the team, the greater the time required for mutual coordination of decisions. Jeffery (1987), (Jeffery et al., 2000) examines dependence of productivity upon the code line numbers and maximum number of team members. He suggests following model form.

$$\text{Productivity} = a \text{ LOC}^b \text{ Maximal Team Member Number}^{-c},$$  \hspace{1cm} (6)

where productivity is defined as number of lines divided by effort expressed in man-months.

Conte, Dunsmore i Shen (1989) introduce COPMO model, which also connects effort, value and number of personnel members. This model is based on a presumption that the total effort required for system development can be divided into individual team member effort plus effort required for coordination of their labour. The derived model has following form:

$$\text{Effort} = a + b \text{ LOC} + c \text{ Average Team member Number}^d.$$  \hspace{1cm} (7)

where average team member number is calculated as quotient of effort and total project duration.

Project management, development team and users often need to estimate total project duration time from the beginning to the very end, or minimum time required for the completion of the system. System creation time obviously depends on the number of the engaged personnel. However, Brooks (1975) rightly pointed out that time dependence upon personnel number is not linear. This fact poses new questions to researchers: is there an optimum number of people working on a project, from productivity point of view? Is it possible to engage extra personnel to work on a project in order to reduce the creation time, at the expense of reduced productivity? At what point does this strategy cease to be achievable?

In order to answer these questions, researchers have developed a number of models connecting effort and time required to create a project. Putnam (1978) (Putnam et al., 2003) suggests a model which connects time required to deliver a system and total effort of lifecycle and the size of the system:

$$\text{LOC} = C_k K^{1/3} t_d^{4/3},$$  \hspace{1cm} (8)

where $\text{LOC}$ is number of code lines, $K$ total effort of lifecycle, and $t_d$ time until the system delivery. $C_k$ is factor whose value depends on the project features. Typical values can be from 2000 in respect of poor software environments with no developed methodology or tools or the like, up to 11000 in respect of mature, developing environments.

Putnam’s model presumes that the effort invested by the staff over the time follows Rayleigh’s curve. This presumption is based on earlier empirical researches (Norden 1963). The model has been derived from the distribution of work obtained in respect of large projects (total effort the size of 30 man-year), with the possibility of extrapolation to smaller projects. Parameter $t_d$ is, in fact, time required to reach the maximum of Rayleigh’s curve. The model also relies on Putnam’s empirical researches, from which he concluded that $K / t_d$ ratio can only have certain discreet values.

Parr (1980) introduced a variation of Putnam’s model. He substituted Rayleigh’s distribution by similar one which does not intersects the coordinate. This substitution has been introduced in order to model the work of system analyzers, who were not taken into account by Putnam.

COCOMO (CONstructive COst Model) represents, in fact, a hierarchy of 3 estimation models, suggested by Boehm (1981), (Boehm et al., 2004) The models range from that of macro estimation of measuring as product value function, to that of macro estimation with structure analysis and 3 levels of multiplier set of phase sensitivity for every attribute of leading expense. Primary motivation for the creation of COCOMO model was to help people realize the cost of the consequences of the decisions they would make in executing, developing and supporting software product. Besides, it enables the estimate of software expenses. Following is the model hierarchy:
Model 1. Basic COCOMO is static single-variable model which calculates software development effort and costs as a function of programme size expressed in estimated LOC.

Model 2. Intermediate COCOMO model estimates effort as a function of programme size and set of "cost drivers" that include subjective assessment of four attribute groups (product, hardware, personnel and project).

Model 3. Advanced COCOMO incorporates all characteristics of the intermediate model, with the addition of "cost drivers" which are evaluated for every phase in the software development (e.g. analysis, design, etc) individually.

Equation of the basic COCOMO model has following form:

\[
\text{Effort} = a \cdot \text{LOC}^b, \quad (9)
\]

\[
\text{NominalTime} = c \cdot \text{Effort}^d. \quad (10)
\]

Finally, it should be said that this, undoubtedly relevant estimation model, has no specifically high accuracy. Being aware of this fact, the designer of the model himself, B.Boehm, says: "At today’s level, the software development model of cost estimation serves the purpose if it can estimate the costs within 20% of real costs, 70% time, and within project class on which it has been calibrated. This is not as accurate as we would like it to be, but it is accurate enough to ensure considerable assistance with economic analysis of software projects and decision making.”

3. EMPIRIC NON-PARAMETRIC ESTIMATION MODEL

It is characteristic for empiric non-parametric models that they use data on projects realized earlier. However, the estimation is not done by applying given mathematic formula but by means of other approaches. Out of these models mentioned herein will be: optimized set reduction technique (OSR), decision-making trunk and neural networks.

Briand, L., Basili, V.R., and Thomas, W (1992), (Briand et al.,1999). OSR selects subset of projects based on which it estimates productivity of the new project. Productivity is defined as effort in man-months divided by the number of code lines. Projects grouped in optimum subset should have similar cost factors, like the new project. For example, all projects are of medium complexity, they have little reliability requirements and large databases. OSR takes the values of cost factors in such a way that the distribution of productivity in respect of the subset of selected projects can be good, in keeping with the introduced statistic criteria. The distribution of probability in respect of productivity is derived from the distribution of frequency of the selected projects above the volume of productivity interval. Productivity in respect of the new project is estimated by calculating the expected value on the basis of the derived probability distribution. Briand, Basili and Thomas, compared the accuracy of OSR technique and COCOMO model, and obtained results which indicated more accuracy in respect of OSR estimation model.

Significant OSR advantage is that it can be applied even with incomplete input data, i.e. when only subset value of the cost factor is known. Srinivasan and Fisher (1995) describe another two non-parametric methods for generating effort estimation model. The first method uses self-learning algorithm to obtain decision-making tree. The other method relies on neural networks. The neural networks model shows smaller mean error than the decision-making tree model. However, training of neural networks is often strenuous. Accuracy of these models is similar to that of the OSR. In order that these models can be applied in practice, calibration should be done on a great number of data, since these models have a great number of independently variable values.

4. EXPERT ESTIMATES

These models are based on consultation of one or more people considered to be experts in software development. For coordination of differing opinions among estimators, often used is one of formal techniques like Delphi. There exist a number of Delphi technique forms. Wideband Delphi ( )encourages those involved to discuss the problem among themselves. This technique is implemented in following steps:

1. Coordinator will acquaint every expert with project specifications and estimation manner.
2. Coordinator will call a meeting of experts to discuss the issues related to the value to be estimated.
3. Each expert will individually and independently complete the form.
4. Coordinator will call another meeting to discuss mainly the estimates that differ most from the others.
5. Experts will complete the forms again.
6. Steps 4 and 5 will be repeated until consensus has been reached.
5. ANALOGUE ESTIMATION MODELS

It is characteristic for these models that in order for estimations to be made, analogies are used between the new project and some of the already completed ones. Comparisons are made between the suggested project and similar projects for which data in respect of cost, time and effort are known. These models require as much data as possible concerning implemented projects. In some aspects, this approach is systematized expert estimation model form, since it is usually experts who decide what projects should the new project be compared with. These techniques require determination of those project characteristics that will be looked for as similar between this and other projects. Effort, time, cost values of these projects are used when making estimates concerning the new project. Basic difficulties with this approach concern identification of projects similar enough, on the basis of which estimates are to be made. Two best known analogue models are ESTOR Mukhopadhyay, T., (1992), and ANGEL Shepperd, (1996), (Shepperd et al., 2000). ESTOR is case-based reasoning model. This case-base reasoning form consists of 5 basic processes:
- Target case specification;
- Search for adequate case to serve as original analogy;
- Transfer solution from the source to the target case;
- Find the differences between the original and the target case;
- Adjust the initial solutions based on the differences found.
With ESTOR, cases are software projects, each of them being represented by a series of metric values. Metrics used by ESTOR are components of function points and input values in the intermediate COCOMO model. ESTOR requires original analogy as per function point value components of target projects. The closest project is identified by means of vector calculation, as smallest distance from the source project. Analogue project effort is taken as initial solution for effort estimation. Differences between analogue and new project are determined by comparing their metrics.
ANGEL has been based on the generalization of the approach by Atkinson and Shepperd (1994), Shepperd M., Jorgensen M., (2007). According to this approach, projects are presented by means of function point components. Analogue projects are neighbours of the new project, and they are reached by calculating vector distance from the new project. Effort concerning the new project is estimated on the basis of the mean effort value in respect of the neighbouring projects.
With ANGEL, user himself determines metrics needed for identification of analogue projects. ANGEL can also automatically determine the optimum metrics subset on the basis of the given data. It can search for 1, 2 or 3 analogue projects and calculate mean value of their efforts.
As one can see, ESTOR and ANGEL have many common features: in both cases, projects are represented by means of easily obtainable metrics, and analogue projects in both cases are identified by calculating vector distance. ESTOR uses only one analogue to determine the estimate, while ANGEL’s estimate can be based on several analogues.
The advantage of analogue estimation models over the empirical parametric models is in their successful application in the cases where valid statistic data dependence can not be determined. Schofield and Kitchenham (1996), (Schofield et al., 2000), give an example of a set of eight projects for which ANGEL gives estimate with mean relative 60% error, while regression linear model gives 226% error.

6. "DOWNWARD" ESTIMATES

Estimation of total effort is made on the basis of the software product global characteristics Shooman M.L., (1996). This estimate is usually based on previous projects and takes into account effort in respect of all function projects. Total effort is then distributed as per components.

7. "UPWARD" ESTIMATES

In this case estimation is made in respect of every project component individually, and total effort is calculated as addition of individual efforts Shooman M.L., (1996). Quite often such approach leaves many global effort components overlooked, such as those linked with integration, system testing and project management.

8. CONCLUSION

In the past four decades a great number of different models and effort estimation methods have been developed. This clearly indicates the awareness among the researchers of the need to improve effort estimation in software engineering. Unfortunately, the fact remains that even though, all the effort invested by the researchers yielded no result as they wished for and, even today, effort estimation still
remains rather unreliable. Whenever estimations are made one actually looks to the future, and naturally, accepts certain level of uncertainty and risk. Risks happen as result of insufficient information, which we can not know in advance. This is the point where good historic background becomes revealed. Risk is measured by uncertainty level in quantitative estimates of resources, costs and distribution.

One should not expect effort estimation to be ever an exact science. Many factors have impact on the software development process. These factors can be human, technical but also political and their impact can never be fully predicted. This should no way be understood as a call to give up estimating because even insufficiently accurate estimates are far better than none.

REFERENCES


