ESE - Enhanced Software Estimation

Luca Santillo (luca.santillo@gmail.com, www.agilemetrics.it)

Introduction

Software estimation research to date offers several alternative methods for forecasting effort, duration, and costs of software projects. In most of them, the major role is played by the size-effort relationship, while costs and duration forecasts are provided through subsequent steps, based on sound project-management structured methods. (Nevertheless, direct statistical regression can be found between project size and duration, average staff, and so on.)

Each method has positive elements and drawbacks. Particularly, the industry environment deplores the possibly large inaccuracy of the results of the various estimation methods, often leading to large deviations between the contract effort and/or cost and the actual values at the end of the project. Such deviations — in practice often justified by some heuristic, sometimes case by case, or just refunded by means of some financial or executive *ad hoc* mechanism — cannot be removed entirely, by definition of "estimation," but they can be strongly reduced by means of a combination of each estimation method's positive elements.

This chapter depicts the fundamental integration of various software estimation methods, usually used separately and as alternatives, and the differentiation of some of their parameters to provide a more accurate and more realistic effort forecast. Together with the theoretical picture of such an enhanced software estimation (ESE) method, a structured approach is proposed for selecting basic parameters and for assigning their coefficients in the specific context.

The Basic Model: IFPUG Guidelines and ISBSG Benchmark

The software estimation method proposed in the IFPUG *Guidelines to Software Measurement* ([IFPUG 2001]) correlates the global project functional size, expressed in function points, with the work effort, expressed in person hours, required to implement the project. More precisely, the IFPUG *Guidelines* propose to "convert the calculated size to effort hours using normalized delivery rates. Project histories yield the normalized delivery rate of function points per hour […]:"

$$PWE = PDR \cdot Size \qquad (0.1)$$

where *PWE* is the estimated project work effort in person hours, *Size* is the functional size of the project in function point, and *PDR* is the expected project delivery rate, given in person hours per function point (sometimes the reciprocal of this number is used, expressed, obviously, in function point per person hour). A further step is then proposed to convert the estimated effort hours to labor cost, using dollars per hour and including "other costs" such as equipment, training, and so on. The proposed sequence should be reiterated at each project life-cycle phase by using the more detailed requirements to recalculate size, estimated effort hours, labor cost, and total estimated cost.

As stated, the PDR should be calculated from previous projects. The IFPUG *Guidelines* specify that "the normalized delivery rates will vary depending on the *project profile*." To normalize delivery rates, a set of Application and Project Attributes to collect and analyze is suggested (see Table 24–1). Attributes should then be used to identify projects with similar characteristics so that appropriate comparisons can be made and normalized delivery rates can be derived.

Table 24–1: Application and Project Attributes (IFPUG Guidelines to Software Measurement)

General Project Characteristics	Resources	Process and Project Management	Technology
Project type	Technical experience	Methodology	Database management systems
Project characteristics	Business experience in functional area	Project management approach	Number of database management systems
Architecture	User	Modeling techniques	Development platform
Degree of innovation	Support	Standards used	Physical environment
Relative project complexity	Software developer	Percentage of reusable code	Testing and debugging tools
System performance requirements	Training	Release strategy	Automated testing tools
Project team		Project structure	Code analysis tools
Organization			Configuration management tools
			Development language(s)
			Operating system(s)
			Communication requirements
			Case tools

Some of these suggestions, in a global framework, have been taken by the International Software Benchmarking Standards Group (ISBSG), which collects and publishes the so-called *benchmark.* The ISBSG found a strong correlation between Size and PWE, adopting the power formula:

$$PWE = A_{regr} \cdot Size^{B_{regr}} \quad (0.2)$$

Equivalently, we can express the same kind of correlation between PDR and Size:

$$PDR = PWE / Size = A_{regr} \cdot Size^{B_{regr}} / Size = A_{regr} \cdot Size^{B_{regr}-1} \quad (0.3)$$

Equations (1.2) and (1.3) include the regression parameters A_{regr} and B_{regr}, which are derived by the best fit of the benchmarking data. The data can be filtered on several project attributes, but once we choose these filters, the values of A_{regr} and B_{regr} are derived solely on a statistical regression basis, with no intrinsic rationale apart from some heuristic explanation of their actual value [ISBSG 1998].

The most recent results from ISBSG [ISBSG 2001] show that the most impacting factors in the correlation analysis are platform and primary programming language. Other factors, such as business area, application type, and CASE tools, affect the PDR, too, but their influence is lower than the first two.

To Adjust or Not To Adjust: The VAF Issue

The current IFPUG approach to software size measurement includes calculation of the value adjustment factor (VAF) by the addition of 14 general system characteristics, each of which has a possible weight of zero to five. As noted by Zuse (ZUSE 1998), these fourteen factors are confusing: they introduce some technical adjustment on the unadjusted function point count, but their scale can result in some not-normalized result. Moreover, the resulting size is no longer a "functional size."

According to these considerations and in order to propose the FP method as a candidate to the ISO working Group devoted to the definition of a standard for software functional measurement, IFPUG has to consider the possible use of unadjusted function point as the main, merely functional software size. Therefore, we should use the unadjusted function point count as the primary effort driver in the estimation process. This does not mean that the 14 general characteristics are not to be considered, but rather that their influence is to be taken into account in some other way during the whole effort estimation process, not only on size, because the average effort of software development derives from technical or quality characteristics beyond mere size.

Internal Benchmarking: The "Best of" Issue

The repository on which the ISBSG benchmark is calculated represents the best part of the software industry (probably the "best 25%"). When using ISBSG results, therefore, we should be aware that the true average values of some or all the parameters could be different from those published by ISBSG. A single organization should perform its own local benchmarking to achieve a realistic analysis of its capabilities and to use it for further estimations.

This calibration of statistical as well as *a priori* estimation parameters and their equations is always required in order to reduce errors in the estimation. Regression from a global benchmark is a good macroeconomic positioning tool, but it can be very inexpedient as an estimation tool if we are not aware of the differences between our project and the globally average project.

An Algorithmic Model: Constructive Cost Model

COCOMO (Constructive Cost Model) by Boehm (COCOMOII 1997)is the most famous algorithmic model for effort (and time) estimation for software projects, since the late 1970s. Here we briefly describe the basic model in its most recent form, called COCOMOII.

The first step of COCOMOII is the estimation of the unadjusted, or nominal project work effort:

$$PWE_{nom} = A \cdot Size^{B} \qquad (0.4)$$

Note the apparent similarity to the function fitted by the ISBSG; in particular, here A is a constant statistically determined, as in the ISBSG regression model. Still there are some remarkable differences:

- **Size is measured in source lines of code (LOC).** Actually, COCOMOII allows us to use function points and then converts them to LOC with the so-called backfiring approach (see also *"FP & LOC: The Backfiring Issue"*).
- **The exponent *B* is calculated with some rationale through some scaling drivers.** The assignments of these scaling drivers can strongly affect the precision of the overall estimation result because of the position of *B* as an exponent (see also *"How sure are we: The Uncertainty Issue"*).

The second step of the basic COCOMOII is the adjustment of the nominal PWE_{nom} to the final *PWE*, by multiplying with the cost drivers of the project (CD_i). The number of these factors, depending on the phase in which we are conducting our estimation, is 7 (Early Design Model) or is "exploded" into 17 (Post-Architecture Model):

$$PWE = PWE_{nom} \cdot \prod_i CD_i = A \cdot Size^B \cdot \prod_i CD_i \quad (0.5)$$

The cost drivers are project attributes such as Required Reuse, Platform Difficulty, Personnel Experience, and Schedule; they represent context-specific considerations. Each *CD* can be a number larger or smaller than 1, so that the overall adjustment of the nominal effort can strongly reduce or increase the nominal effort estimation, depending on the values of the actual project attributes. The value of each factor is assigned with the help of a nominal scale from Very Low to Extra High; the Nominal (neutral) value is exactly 1 so that it does not influence the adjustment. The use of nominal versus ordinal scales in different method variants can be faced with the help of fuzzy logic concepts.

As for benchmarking, the scale values of each COCOMO factor should be calibrated to reflect the local situation in each organization environment.

FP & LOC: The Backfiring Issue

The backfiring approach employs the relation between physical size (expressed in lines of code) and effort, determined statistically, depending on the programming language used, or the language level, and assumes a bond between functional size (FP) and physical size (LOC). This bond has been strongly criticized because of the different dimensions (attributes) measured by the two metrics: LOC and FP are not equivalent but rather complementary metrics.

Conversion of LOC to FP (and vice versa) should be avoided, because it can introduce the strongest error in the estimation process.

Parametric Models: The Simplicity versus Differentiation Issue

Evaluating size alone to estimate effort would be good. This would be easy and quick, but both the benchmarking by ISBSG and the algorithmic model of COCOMO show that it's very risky not to take into account several "adjustment" factors to differentiate the given project from the average project in the sample used as a basis for the estimates.

Of course, every additional parameter in the chosen model needs methods, metrics, and time to be evaluated. This means every parameter can add possible errors in the estimation process (see also "How Sure Are We: The Uncertainty Issue").

We should at least filter the initial benchmarking database by some main categories, such as project type (development or enhancement) and main programming language level, and use at least some of the adjustment factors proposed by any algorithmic model, such as COCOMO. Overusing this filtering process in search of the most similar projects usually results in a sample too small to provide significant statistics: as every project manager knows, "each project is different." The statistical sample to start from should be much larger than the number of freedom degrees of the studied relationship corresponding to the possible influencing project attributes.

In the research field, there are some evidences that, although the list of factors impacting the size-effort relationship is not fixed for every environment and business area, the number of these factors should be not large (10–15 independent factors work well). Many factors could be identified, but only some of them have a strong influence on the final effort for each project. This is especially true when an organization establishes its own software metrics databases in addition to benchmarking its data against that of other companies. In this case, accurate effort estimation is possible with just a very small number of productivity factors.

For each product area, experience can show which factors to include and which to exclude for each new project in that area. A way to extract and prioritize such relevant factors from a larger set of possible attributes is the Analytic Hierarchy Process (AHP), a quantification technique of expert judgments.

AHP, originally proposed by Saaty in [SAATY 1981] and then recalled several times in the software engineering field, provides a means of making decisions or choices among *n* homogeneous alternatives. The method requires pair-wise comparisons to be made between each two factors with respect to their importance in the effort influence impact. Comparisons are made by posing the question, "Of two factors *i* and *j*, which is more important and how much more?" The strength of preference is usually expressed on a ratio scale of 1:9. A preference of 1 indicates equality between two factors, while a preference of 9 (absolute importance) indicates that a factor is nine times more important than the one to which is being compared.

The pair-wise comparisons result in a *reciprocal n*-by-*n matrix M*, where $m_{ii} = 1$ (on the diagonal) and $m_{ji} = 1/ m_{ij}$ (*reciprocity property*, assuming that if factor *i* is "*x*-times" more important than factor *j*, then factor *j* is "1/*x*-times" more important, or equally "*x*-times" less important than factor *i*). Given this comparison matrix, we can recover the prioritization scale of the *n* factors; a quick way to do that is to normalize each column in *M* and then average the values across the rows: this "average column" is the normalized vector of weights (or priorities) of the *n* factors. It is also possible to compute a consistency index based on the properties of the matrix *M* to verify the goodness of the judgments.

The weights of relative importance of the factors can then be used to select only the first ones, or more exhaustively to use all of them, but to compare the errors in their evaluation with respect to their importance for influencing the effort estimation.

The Software Reuse Issue

Software reuse is the process of creating software systems from predefined software components. A reusable component may be code, but at all levels of development —

from requirements specifications to code — there are components that by the nature of implementing tasks and representing information on a computer must appear over and over in software applications: requirements, specifications, designs, test cases, data, prototypes, plans, documentation, frameworks, templates, and so on ([McClure 1995]). Many studies demonstrate that software reuse can cut software development time and costs; moreover, when software is reused multiple times, the defects fixed in each reuse accumulate, resulting in higher quality. Generally, reuse is to be considered among the most important factors impacting productivity.

From the Rine and Nada research surveys ([RINE 1998, NADA 1998]), for example, we find that, out of about one hundred projects:

- 57% of the projects realized high commonality with the requirements of previous project(s) at the requirements phase.
- 43% of the projects realized high commonality with the design of previous project(s) at the design phase.
- 38% of the projects realized high commonality with the code (documentation) of previous project(s).

Rine and Nada, in their Reuse Reference Model (RRM), propose five reuse levels as a reuse global attribute of a software project (see Table 24–2).

Table 24–2: Reuse Reference Model Levels

Abbreviation	Level	Nominal Value	Percent Commonality
RL1	Level 1	Very Low	0–20%
RL2	Level 2	Low	21–40%
RL3	Level 3	Average	41–60%
RL4	Level 4	High	61–80%
RL5	Level 5	Very High	>80%

The FP size of a software project is calculated according to the IFPUG current rules, without taking into account software reuse. The obtained number is the size of actual functionality provided to the user (note that the general system characteristic called *Reusability*, in the VAF evaluation, if used, is meant to increase the size when there are explicit requirements of reuse in future projects; it's not a measure of a component's past reuse in the current project).

In the COCOMOII model, reuse is more explicitly taken into account: it can be used as a percentage to adjust the size *and* is also one of the 17 factors used to adjust the effort estimation. We therefore propose to apply the reuse level at the size stage, adjusting the measured size of the project to represent the effective size to be worked (developed or maintained). The reuse percentage may consist of, or may be derived from, the following categories (with total equal to 100 percent):

- Reused percentage of developed components from other projects
- Reused percentage of commercial off-the-shelf components
- Developed percentage of components for reuse by other projects
- Developed percentage of components unique to the identified projects

This can be considered the zero-order precision reuse measurement. A more efficient way to take reuse into account would be specific measurement of the reuse adjustment of each measured component (that is, of each ILF, EIF, EI, EO, and EQ counted for the studied project). For example, in [ABRAN 1995] we find that "there is an interesting mapping between the EIF definition and the conventional interpretation of data reused 'as is,' without modification, which is called 'black-box reuse.' We can then say that an EIF is a reuse logical file." For each counted function, we should define a reuse percentage, based on similarities with other functions in the same project or similar projects in the same area. The similarity can be estimated or based on common data element types, record element types, or file types referenced, as proposed by the Netherlands FP users group ([NESMA 2000]).

Note that reusing has a cost of its own that should be taken into account at the project level as well as at the organization level. Moreover, not having a benchmark of current organizational practices that enables us to determine what effort is actually avoided through reuse could lead to a misleading percentage reuse metric.

The Intrinsic Complexity Issue

One of the main reasons for the initial resistance to the introduction of functional measurement methods in the industry has been the fact that the intrinsic (read "algorithmic") complexity of software systems seemed not enough taken into account. This led to the introduction of some general system characteristics, such as complex processing, in the VAF factor of the IFPUG method, or the CPLX adjustment factor of the COCOMOII estimation model. Apart from general discussion about applying nonfunctional factors to the size (the VAF issue), we should look for a more accurate method to include this factor in the effort-estimation model. As for software reuse, we can think of a granular adjustment of the FP number at each element level, to achieve a second effective size next to the reuse-adjusted size. This adjustment can be obtained by applying a coefficient, usually in the interval (0, 2), to the number of FP of each element. Several guidelines and proposed classes can be found in the literature to help choose these complexity values.

Requirements Volatility: The Change Requests Issue

The IFPUG *Guidelines*, regarding the scope change, state that

> "… the later in a project change is introduced, the more costly it is to implement. Introducing a change early in the project, such as in requirements analysis, causes little additional work. A requirements change during construction may require significant changes to the architecture, design, and previously tested and approved components causing large amounts of rework and increasing the cost. When estimating changes to a system or project, it is important that the impact of the change is fully assessed and the size is correctly estimated."

The quantity of scope changes is therefore another important factor that can strongly influence the true value of effort, cost, and time of the project. During the project, we should trace the change requests to analyze their impact. Although the COCOMO model considers requirements volatility (RVOL) to be a global adjustment factor of the

estimation, a more granular, quantitative approach (related to the FP measurement method) has been proposed recently by Meli [MELI 2001].

How Sure Are We: The Uncertainty Issue

The IFPUG *Guidelines* recognize the uncertainty of estimates. Because the scope of the project is not fully defined early, the size is not accurately known. Different values estimated for the size provide different estimates for the effort, of course, but this fact does not provide in itself any consideration of the precision of a single estimate. Any estimation model cannot be seriously performed without consideration of its possible deviations between estimated and true values.

COCOMOII proposes some simple ranges for *optimistic* and *pessimistic* estimates as half the original estimate or twice the original estimate, but this range can be determined more accurately with the following approach. Let's apply the method the physical sciences have used for centuries when performing measurements: error analysis, or the *theory of measurement error propagation* [TAYLOR 1982].

Let the function $y = y(x_1, \ldots, x_N)$ be the model we use to estimate the dependent variable y from the N independent variables $x_i, \ldots, x_N$, and let Δx_i be the estimated error on the value measured for each variable x_i. That is, the true value of x_i belongs to the interval $(x_i \pm \Delta x_i)$ for each i from 1 to N. Then the true value of y belongs to the interval $[y(x_1, \ldots, x_N) \pm \Delta y]$, where Δy is given from the derivatives formula:

$$\Delta y = \sqrt{\sum_{i=1}^{N}\left[\left(\frac{\partial y}{\partial x_i}\Delta x_i\right)^2\right]} = \sqrt{\left(\frac{\partial y}{\partial x_1}\Delta x_1\right)^2 + \ldots + \left(\frac{\partial y}{\partial x_N}\Delta x_N\right)^2} \qquad (0.6)$$

Let's see an application example. To derive the development effort y (in person hours) to the size x (in function points) of a software program, we can use the function:

$$y = A \cdot x^B \qquad (0.7)$$

For this example, consider $A = 10 \pm 1$, $B = 1.10 \pm 0.01$ (the proposed values are only valid for this example and should not be applied to any actual case). Note that having fixed statistical values for some parameters in our model does not mean necessarily that these values are exact: their associated errors can be derived from the standard deviation of the statistical sample from the fitting function y. Depending on the reliability of the sample, such errors could be considered significant or not.

To evaluate the error on y, given the errors on the parameters A and B and on the independent variable x, we have to perform some calculus, recalling that

$$\frac{\partial}{\partial x}\left(A \cdot x^B\right) = A \cdot B \cdot x^{B-1}, \quad \frac{\partial}{\partial A}\left(A \cdot x^B\right) = x^B, \quad \frac{\partial}{\partial B}\left(A \cdot x^B\right) = A \cdot x^B \cdot \ln x \qquad (0.8)$$

We then apply some measurement process and obtain for x the estimated value of 1,000 ± 200 FP, or a possible uncertainty of 20 percent. Collecting all data and applying the equation [1.1] to this example, we then obtain:

$$y = A \cdot x^B = 10 \cdot 1{,}000^{1.10} = 19{,}952.6 \text{ (person hours)} \qquad (0.9)$$

$$\Delta y = \sqrt{\left[\left(A \cdot B \cdot x^{B-1}\right)\Delta x\right]^2 + \left[\left(x^B\right)\Delta A\right]^2 + \left[\left(A \cdot x^B \cdot \ln x\right)\Delta B\right]^2} =$$

$$= \sqrt{\left[21.948 \times 200\right]^2 + \left[1,995.262 \times 1\right]^2 + \left[137,827.838 \times 0.01\right]^2} = 5,015.9 \text{ (person hours)}$$

(0.10)

Taking only the first significant digit of the error, we obtain for *y* the estimated range 20,000 ± 5000, or a possible uncertainty of 25 percent. Note that the percent error on *y* is not the simple sum of the percent errors on *A*, *B*, and *x*, because the function assumed in this example is not linear.

A similar calculation, on the second step of the COCOMOII model, yields:

$$y_{adj} = y_{nom} \cdot \prod_i f_i \qquad (0.11)$$

$$\frac{\partial}{\partial y_{nom}}\left(y_{nom} \cdot \prod_i f_i\right) = \prod_i f_i, \quad \frac{\partial}{\partial f_j}\left(y_{nom} \cdot \prod_i f_i\right) = y_{nom} \cdot \frac{\prod_i f_i}{f_j} \qquad (0.12)$$

If, for example, we have y_{nom} = 20,000 ± 5000, and 7 factors f_i, for each of which (for simplicity of the example) we assume the same value = 0.95 ± 0.05), then:

$$y_{adj} = y_{nom} \cdot \prod_i f_i = 20,000 \cdot \prod_1^7 0.95 = 20,000 \cdot 0.95^7 = 13,966.7 \qquad (0.13)$$

$$\Delta y_{adj} = \sqrt{\left[\left(\prod_i f_i\right)\Delta y_{nom}\right]^2 + \left[\left(y_{nom} \cdot \frac{\prod_i f_i}{f_1}\right)\Delta f_1\right]^2 + \ldots + \left[\left(y_{nom} \cdot \frac{\prod_i f_i}{f_7}\right)\Delta f_7\right]^2} =$$

$$= \left(\prod_i f_i\right)\sqrt{\left(\Delta y_{nom}\right)^2 + \left(y_{nom}\right)^2\left(\frac{\Delta f_1}{f_1} + \ldots + \frac{\Delta f_7}{f_7}\right)^2} =$$

$$= 0.95^7 \cdot \sqrt{\left(5,000\right)^2 + \left(20,000\right)^2\left(\frac{0.05}{0.95} + \ldots + \frac{0.05}{0.95}\right)^2} = 5,146.8 \pm 5000 \qquad (0.14)$$

We therefore see that each new factor in the estimation process can apparently make the estimation more accurate (the y_{adj} is reduced with respect to its nominal value because all the factors are smaller than 1), but its percent error is increased (it's now about 36 percent, versus the 25 percent of the nominal estimate).

Case by case we should decide whether to accept the measured value of each factor in the model, to refine it (reducing its error), or to completely avoid using that factor in the overall model.

Enhanced Software Estimation: An Integration

Enhanced Software Estimation (ESE), release 2001, is based on integration of the previously depicted approaches: benchmarking statistical results and the constructive cost model, with considerations about the specific issues we have addressed (reuse, change request, and error analysis).

ESE can be shortly illustrated as a practical recipe:

1. Prepare a local benchmark data set of historical projects (otherwise, get an external, global data set, such as the ISBSG repository).
2. Filter the benchmark data set to match as much as possible the profile of the new project; do not overuse filtering.
3. Calculate the nominal effort regression equation from the selected subset, using unadjusted FP as the size quantity:

$$PWE_{nom} = A_{regr} \cdot UFP^{B_{regr}} \qquad (0.15)$$

4. Measure or estimate the size of the new project (UFP) and evaluate its general system characteristics.
5. Adjust the size from point 4 (UFP) with intrinsic complexity, function by function if possible (unless you used intrinsic complexity explicitly to filter the benchmark data set). Call this number XCUFP (extended by complexity unadjusted FP).
6. Adjust the size from point 5 with reuse percent metric, function by function if possible (unless you used software reuse explicitly to filter the benchmark data set). Call this number XCRUFP (extended by complexity and reuse unadjusted FP).
7. Calculate the nominal effort estimate by applying equation (1.15) to the size measure from point 6 (XCRUFP):

$$PWE_{nom} = A_{regr} \cdot XCRUFP^{B_{regr}} \quad (0.16)$$

8. Consider a set of possible cost drivers, starting from the COCOMO list, the VAF list, and such. Exclude every factor that has already been used when filtering the benchmark data set (point 2) and/or cannot be adequately measured or estimated for the new project.
9. Prioritize the factors extracted at point 8.
10. For the factors extracted at point 8, validate the calibration of values to the new project domain. Pay maximum attention to the values of the factors that come first in the prioritization order from point 9.
11. Evaluate the factors extracted at point 8 and adjust by multiplication the nominal effort estimation from point 7:

$$PWE_{adj} = PWE_{nom} \cdot \prod_{selection} EM_{selected} \qquad (0.17)$$

12. Calculate error propagation on the output estimate, from the estimated errors on the input estimates and factors, as depicted in the section *"How Sure Are We: The Uncertainty Issue"*.
13. Trace and measure change requests during the project.
14. Reiterate the estimation process during the project to refine the estimates and/or to take into account any change request.

Future Enhancements

A lot of issues still need to be faced, included, or solved in the actual ESE 2001 model. For example:

- ESE takes FP as an alternative measure for software projects, instead of the original LOC measure from the COCOMO model. Recent research trends show that these metrics are still both useful for different kinds of estimation issues. An integration of these and other possible metrics, as depicted by Meli [MELI 2001], is expected to provide better estimation results.
- Integration between estimation models, such as benchmarking statistics, the COCOMO model, and others, has to be performed, carefully avoiding duplication or partial overlap in productivity factors. For example, the reuse factor of any estimation model can overlap with other factors, such as team experience, quality, and so on.
- FP are often estimated, not counted, due to a lack of precision in the requirement elicitation. Several ways to estimate FP exist; one of the more reliable is Early & Quick Function Point Analysis (EQFPA) illustrated in the corresponding chapter of this book.
- The reuse issue itself deserves much more attention. Structured reuse metrics are required, and reuse impact on effort estimation is the subject of future research.
- ESE is a sequential model, from (estimated) size to estimated effort; from this value, we can then obtain estimate costs and time forecasts for the project. Sound statistics show how these factors are strongly interrelated so that *schedule compression*, for example, can increase the effort required in a nonlinear way, while decreasing quality. A recent trend is to consider all project attributes to be linked by Bayesian probabilistic inference (one of the first proposals is by Fenton, in [FENTON 1999]).

Solving such issues is the goal of future research, and the possible solutions will be included in future releases of ESE.

References

[ABRAN 1995] Abran, A., Desharnais, J., *Measurement of functional reuse in maintenance*, Journal of Software Maintenance: Research and Practice, vol. 7, no. 4, 1995.

[COCOMOII 1997] Boehm, B., et al., *COCOMO II Model Definition Manual*, Version 1.4, University of Southern California, 1997.

[FENTON 1999] Fenton, N., Neil, M., *Software Metrics and Risk*, FESMA 99 Proceedings, Amsterdam, 1999.

[IFPUG 2001] - International Function Point Users Group, *Guidelines to Software Measurement*, Version 1.1, IFPUG, www.ifpug.org, 2001.

[ISBSG 1998] Milne, B.J., Luxford, K.B.G., *Worlwide Software Development – The Benchmark*, Release 5, International Software Benchmarking Standards Group, 1998.

[ISBSG 2001] Hill, P.R., Practical Project Estimation – A toolkit for estimating software development effort and duration, ISBSG & SEA, 2001.

[LIM 1999] Lim, W.C., *Why the Reuse Percent Metric should never be used alone*, Annual Workshops on Institutionalizing Software Reuse, WISR 9, Austin, 1999.

[MCCLURE 1995] McClure, C., *Model-driven software reuse*, Extended Intelligence, Inc., 1995.

[MELI 2001] Meli, R., Measuring Change Requests to support effective project management practices, ESCOM 2001 Proceedings, London, 2001.

[NADA 1998] Nada, N., Rine, D. C., *A Validated Software Reuse Reference Model Supporting Component-Based Management*, International Workshop on Component-Based Software Engineering, 1998.

[NESMA 2000] Hennie Huijgens, Estimating cost of software maintenance: a real case example, NESMA, www.nesma.nl, 2000.

[RINE 1998] *Software Reuse Manufacturing Reference Model: Development and Validation*, Survey, School of Information Technology and Engineering, George Mason University, Fairfax, Virginia, 1998.

[SAATY 1981] Saaty, T.L., Alexander, J.M., *Thinking with Models*, Pergamon Press, 1981.

[TAYLOR 1982] Taylor, J.R., An Introduction to Error Analysis, The Study of Uncertainties in Physical Measurements, University Science Books, 1982.

[ZUSE 1998] Zuse, H., *A Framework of Software Measurement*, de Gruyter, 1998.