SOFTWARE PROJECT MANAGEMENT

Module-2: PROJECT LIFE CYCLE AND EFFORT ESTIMATION

CSE4016

PRIYANKA SINGH

Basics of Software estimation

- One definition of a successful project is that the system is delivered ‘on time and within budget and with the required quality, which implies that target are set and the project leader then tries to meet those targets.
- Estimating the effort required to implement software is notoriously difficult. Some of the difficulties of estimating are inherent in the very nature of software, especially its complexity and invisibility.
- In addition the intensely human activities that make up system development cannot be treated in a purely mechanistic way.

- Other difficulties include:
  - Novel applications of software
  - Changing technology
  - Lack of homogeneity of project experience

Where are estimates done?

Estimates are carried out at various stages of a software project.

- ***Strategic planning-*** The costs of computerizing potential applications as well as the benefits of doing so might need to be estimated to help divide what priority to give to each project.
- ***Feasibility study*** -This ascertains that the benefits of the potential system will justify the costs.
- ***System specification-*** The *effort needed to implement different design proposals will need to be estimated,* Estimates at the design stage will also confirm that the feasibility study is still valid, taking into account all that has been learnt during detailed requirements analysis.
- **Evaluation of suppliers' proposals**
- **Project planning-** As the planning and implementation of the project progresses to greater levels of detail, more *detailed estimates of smaller work components* will be made.

The basis for software estimating

- ***The need for historical data-*** Nearly all estimating methods need information about how projects have been implemented in the past. However, care needs to be taken in judging the applicability of data to the estimator's own circumstances because of possible differences in environmental factors such as the programming languages used, the software tools available, the standards enforced and the experience of the staff.
- ***Measure of work-*** It is normally not possible to calculate directly the actual cost or time required to implement a project. The time taken to write a program might vary according to the competence or experience of the programmer. Implementation times might also vary because of environmental factors such as the software tools available.

Complexity-

- Two programs with the same KLOC will not necessarily take the same time to write, even if done by the same developer in the same environment.
- One program might be more complex. Because of this, SLOC (Source lines of code) estimates have to be modified to take complexity into account.
- Attempts have been made to find objective measures of complexity, but often it will depend on the subjective judgement of the estimator.

Problems with over- and under-estimates

An over-estimate might cause the project to take longer than it would otherwise. This can be explained by the application of two laws-

- **Parkinson's law-** "Work expands to fill the time available", which implies that given an easy target staff will work less hard.
- **Brook's 1aw-** The effort required to implement a project will go up disproportionately with the number of staff assigned to the project. As the project team grows in size so will the effort that has to go into management, co-ordination and communication.

The danger with the under-estimate is the effect on quality. Staff, particularly those with less experience, might respond to pressing deadlines by producing work which is sub-standard.

Software effort estimation techniques

Barry Boehm, in his classic work on software effort models, identified the main ways of deriving estimates of software development effort as:

- **Algorithmic models** - which use 'effort drivers' representing characteristics of the target system and the implementation environment to predict effort;
- **Expert judgement** - where the advice of knowledgeable staff is solicited;
- **Analogy** - where a similar, completed, project is identified and its actual effort is used as a basis for the new project;
- **Parkinson** - which identifies the staff effort available to do a project and uses that as the 'estimate';
- **Price to win** - where the 'estimate" is a figure that appears to be sufficiently low to win a contract;
- **Top-down** - where an overall estimate is formulated for the whole project and is then broken down into the effort required for component tasks;
- **Bottom-up** - where component tasks are identified and sized and these individual estimates are aggregated.

The top-down approach and parametric models

- The top-down approach is normally associated with parametric (or algorithmic) models. These may be explained using the analogy of estimating the cost of rebuilding a house.
- The effort needed to implement a project will be related mainly to variables associated with characteristics of the final system. The form of the parametric model will normally be one or more formulae in the form:

Effort = (system size) x (productivity rate)

Some parametric models, such as that implied by function points, are focused on system or task size, while others, such are COCOMO, are more concerned with productivity factors.

Estimating by analogy

- The use of analogy is also called case-based reasoning. The estimator seeks out projects that have been completed and that have similar characteristic, to the new project.
- The effort that has been recorded for the matching source case can then be used as a base estimate for the target.
- The estimator should then try to identify any differences between the target and the source and make adjustments to the base estimate for the new project.
- Identify the similarities and differences between the different systems:

$$distance = \text{square-root of } ((target\_parameter_1 - source\_parameter_1)^2 + \ldots + (target\_parameter_n - source\_parameter_n)^2)$$

Example

- Say that the cases are being matched on the basis of two parameters, the number of inputs to and the number of outputs from the system to be built. The new project B is known to require 7 inputs and 15 outputs. One of the past cases, Project A has 8 inputs and 17 outputs. Is Project B a better analogy with the target than Project A?

Exercise

- Project B has 5 inputs and 10 outputs. What would be the Euclidean distance between this project and the target new project being considered above? Is Project B a better analogy with the target than Project A?

COSMIC Full function points

- The COSMIC method defines the principles, rules and a process for measuring a standard functional size of a piece of software.
- ‘Functional size’ is a measure of the amount of functionality provided by the software, completely independent of any technical or quality considerations.

- The COSMIC method may be used to size software such as business applications; real-time software; infrastructure software such as in operating systems; and hybrids of these.
- The common characteristic of all these types of software is that they are dominated by functions that input data, store and retrieve data, and output data.
- Subject to the above, the method may be applied to measure the Functional User Requirements ('FUR') of software:
  - At any level of decomposition, e.g. a 'whole' piece of software or any of its components, sub-components, etc;
  - In any layer of a multi-layer architecture;
  - At any point in the life-cycle of the piece of software;

COCOMO

- Boehm's COCOMO (Constructive COst MOdel) is often referred to in the literature on software project management, particularly in connection with software estimating. The term COCOMO really refers to a group of models.
- COCOMO (Constructive Cost Model) is a combination of parametric estimation equation and weighting method.
- Based on the estimated instructions DSI (Delivered Source Instructions), the effort is calculated by taking into consideration both the attempted quality and the productivity factors.
- COCOMO is based on the conventional top-down programming and concentrates on the number of instructions.

1. Basic COCOMO:

- Basic COCOMO model is static single-valued model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code.
- By means of parametric estimation equations, the development effort and the development duration are calculated.
- In this connection it is differentiated according to system types (organicbatch, semidetached-on-line, embedded-real-time) and project sizes (small, intermediate, medium, large, very large).

2. Intermediate COCOMO:

- Intermediate COCOMO model computes software development effort as a *function of program size and a set of "cost drivers"* that include subjective assessments of product, hardware, personnel, and project attributes.
- The estimation equations are now taking into consideration 15 influence factors; these are *product attributes* (like software reliability, size of the database, complexity), *computer attributes* (like computing time restriction, main memory restriction), *personnel attributes* (like programming and application experience, knowledge of the programming language), and *project attributes* (like software development environment, pressure of development time).

3. Detailed COCOMO:

- Advanced COCOMO model incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step, like analysis, design, etc.

- In this case the breakdown to phases is not realized in percentages but by means of influence factors allocated to the phases.

- At the same time, it is differentiated according to the three levels of the product hierarchy (module, subsystem, system); product-related influence factors are now taken into consideration in the corresponding estimation equations.

COCOMO II-a Parametric Productivity Model:

- ***Basic COCOMO***:

Boehm originally based his models in the late 1970s on a study of 63 projects.

$$\textit{Effort} = a(\textit{size})^b$$

Generally, information systems were regarded as organic while real-time systems were embedded.

where effort is measured in ***pm***, or the number of ***'person-months'*** consisting of units of 152 working hours, size is measured in ***kdsi*** *(kilo Delivered Source Instructions*) , thousands of delivered source code instructions, and ***a*** and ***b*** are constants.

The first step was to derive an estimate of the system size in terms of ***kdsi***. The constants, ***a*** and ***b*** (see Table), depended on whether the system could be classified, in Boehm's terms, as 'organic', 'semi-detached' or 'embedded'.

These related to the technical nature of the system and the development environment.

- ***Organic mode*** - this would typically be the case when relatively small teams developed software in a highly familiar in-house environment and when the system being developed was small and the interface requirements were flexible.

- ***Embedded mode*** - this meant the product being developed had to operate within very light constraints and changes to the system were very costly.

- ***Semi-detached mode*** - this combined elements of the organic and the embedded modes or had characteristics that came between the two.

System type	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

Examples

Suppose size is 200 KLOC, Calculate effort

- Organic
  - $2.4(200)^{1.05} = 626$ staff-months
- Semi-Detached
  - $3.0(200)^{1.12} = 1{,}133$ staff-months
- Embedded
  - $3.6(200)^{1.20} = 2{,}077$ staff-months

Project Duration

$$Development\ time = c(Effort)^d$$

$$Average\ Staff\ Size = Effort/development\ time$$

$$Productivity = Size/Effort$$

Mode	c	d
Organic	2.5	0.38
Semi-detached	2.5	0.35
Embedded	2.5	0.32

Example

- Suppose size is 200 KLOC, Calculate development time?
- Organic
  - TDEV = $2.5(626)^{0.38}$ = 29 months
- Semi-detached
  - TDEV = $2.5(1133)^{0.35}$ = 29 months
- Embedded
  - TDEV = $2.5(2077)^{0.32}$ = 29 months

Complete Example, Organic

Suppose an organic project has 7.5 KLOC,

- Effort $2.4(7.5)^{1.05} = 20$ staff–months
- Development time $2.5(20)^{0.38} = 8$ months
- Average staff 20 / 8 = 2.5 staff
- Productivity 7,500 LOC / 20 staff-months
  = 375 LOC / staff-month

Complete Example, Embedded

Suppose an embedded project has 50 KLOC,

- Effort $3.6(50)^{1.20} = 394$ staff–months
- Development time $2.5(394)^{0.32} = 17$ months
- Average staff 394 / 17 = 23 staff
- Productivity 50,000 LOC / 394 staff-months
  = 127 LOC / staff-month

Intermediate COCOMO

$$Effort = a(size)^b * C$$

Where

- *E is the effort*
- *a and b are constants (as before)*
- *size is thousands of lines of code in KLOC*
- *C is the effort adjustment factor*

- Intermediate COCOMO introduces Cost Drivers.
  - Product Attributes
  - Computer Attributes
  - Personnel Attributes
  - Project Attributes

- They are used because
  - they are statistically significant to the cost of the project; and
  - they are not correlated to the project size (KLOC).

“The models are just there to help, not to make the management decisions for you.”

-- Barry Boehm