Software Project Planning 4

After the finalisation of SRS, we would like to estimate size, cost and development time of the project. Also, in many cases, customer may like to know the cost and development time even prior to finalisation of the SRS. Hence, whether we estimate before SRS or after SRS, it would always be very critical and crucial decision for the project. Estimation of cost and development time are the key issues during project planning. The correlation between cost, development time, and the planning process can best be illustrated by an example.

Suppose we want to put an addition on our home. After deciding what we want and getting several quotations, most of which are around 2.5 lacs, we pick up a builder who offers to do the job in two months for 2.0 lacs. We sign an agreement and the builder starts the work. After about a month into the job, the builder comes and explains that because of the problems the job will take an extra month and cost an additional 0.5 lacs. This creates several problems. First, we badly need space and another month of delay is a real inconvenience. Second, we have already arranged for a loan and do not know from where we can get this additional amount of Rs. 0.5 lac. Third, if we get a lawyer and decide to fight the builder in court, all work on the job will stop for many months while the case is decided. Fourth, it would take a great deal of time and probably cost even more to switch to a new builder in the middle of the job.

On exploration, we conclude that the real problem is that the builder did a sloppy job of planning. Builder might have forgot to include some major costs like the labour or materials to do the woodwork or final plastering and painting. Because other quotations were close to Rs. 2.5 lacs, we know that this is pretty fair price. At this point, we have no option but to try to negotiate a lower price but will continue with the current builder. However, we would neither use this builder again, nor would probably recommend the builder to anyone else [HUMP95].

This is the essential issue in the planning process; being able to make plans that accurately represent what we can do. Business operates on commitments, and commitments require plans. The failure of many large software projects in the 1960s and early 1970s highlighted this problem of poor planning. The delivered software was late, unreliable, costed several times the original estimates and often exhibited poor performance characteristics. These projects did not fail because managers or developers were incompetent. The fault was in the approach of planning that was used. Planning techniques derived from small-scale projects did not scale up to large systems development.

Software managers are responsible for planning and scheduling project development. They supervise the work to ensure that it is carried out to the required standards. They monitor progress to check that the development is on time and within budget. Good managers cannot guarantee project success. However, bad managers usually result in project failure. Usually, the software is delivered late, costs more than originally estimated and fails to meets its re-

quirements [SOMM95]. The project planning must incorporate the major issues like size and cost estimation, scheduling, project monitoring and reviews, personnel selection and evaluation, and risk management.

In order to conduct a successful software project, we must understand [PRES2K]

- scope of work to be done
- the risk to be incurred
- the resources required
- the task to be accomplished
- the cost to be expended
- the schedule to be followed

Software planning begins before technical work starts, continues as the software evolves from concept to reality, and culminates only when the software is retired. The various steps of planning activities are illustrated in Fig. 4.1. As shown, first activity is to estimate the size of the project. The size is the key parameter for the estimation of other activities. It is an input to all costing models for the estimation of cost, development time and schedule for the project. If size estimation is not reasonable, it may have serious impact on the other estimation activities.

Resources requirements are estimated on the basis of cost and development time. Project scheduling may prove to be very useful for controlling and monitoring the progress of the project. This is dependent on the resources and development time.

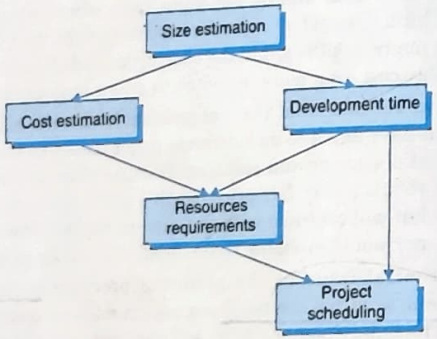

Fig. 4.1: Activities during software project planning

4.1 SIZE ESTIMATION

Some programs are written in C, some in PASCAL, others in FORTRAN, and still others in assembly language. Some programs are for GUI applications and some are for batch processing. Some are written using the latest software engineering techniques, while others are developed without adequate planning. Some programs are well documented with carefully crafted internal comments, other programs are written in a quick and dirty fashion with no comments at all. But, there is one characteristic that all programs share—they all have size [CONT86].

The estimation of size is very critical and difficult area of the project planning. It has been recognised as a crucial step from the very beginning. The difficulties in establishing units for measuring size lie in the fact that the software is essentially abstract: it is difficult to identify the size of a system. Other engineering disciplines have the advantage that a bridge or a building or a road can be seen and touched, they are (sometimes literally) concrete. Many attempts have been made at establishing a unit of measure for size. The more widely known are given below.

4.1.1 Lines of Code (LOC)

This was the first measurement attempted. It has the advantage of being easily recognisable, seen and therefore counted. Although this may seem to be a simple metric that can be counted algorithmically, there is no general agreement about what constitutes a line of code. Early users of lines of code did not include data declarations, comments, or any other lines that did not result in object code. Later users decided to include declarations and other unexecutable statements but still excluded comments and blank lines. The reason for this shift is the recognition that contemporary code can have 50% or more data statements and that bugs occur as often in such statements as in real code. For example, in the function shown in Fig. 4.2, if LOC is simply a count of the number of lines then Fig. 4.2 contains 18 LOC [CONT86].

```
1.  int. sort (int x[ ], int n)
2.  {
3.  int i, j, save, im1;
4.  /* This function sorts array x in ascending order */
5.  If (n < 2) return 1;
6.  for (i = 2; i < = n; i++)
7.  {
8.  im1 = i – 1;
9.  for (j = 1; j < = im; j++)
10. if (x[i] < x[j])
11. {
12. Save = x[i];
13. x[i] = x[j];
14. x[j] = save;
15. }
16. }
17. return 0;
18. }
```

Fig. 4.2: A function for sorting an array in ascending order

But most researchers agree that the LOC metric should not include comments or blank lines. Since these are really internal documentation and their presence or absence does not affect the functions of the program. Moreover, comments and blank lines are not as difficult to

construct as program lines. The inclusion of comments and blank lines in the count may encourage developers to introduce artificially many such lines in project development in order to create the illusion of high productivity, which is normally measured in LOC/PM (lines of code/person-month). When comments and blank lines are ignored, the program in Figs. 4.2 contains 17 LOC.

However, there is a fundamental reason for including comments in the program. The quality of comments materially affects maintenance costs because maintenance person will depend on the comments more than anything else to do the job. Conversely, too many blank lines and comments with poor readability and understandability will increase maintenance effort. The problem with including comments is that we must be able to distinguish between useful and useless comments, and there is no rigorous way to do that. Therefore, it is always advisable not to consider comments and blank lines while counting for LOC.

Furthermore, if the main interest is the size of the program for specific functionality, it may be reasonable to include executable statements. The only executable statements in Fig. 4.2 are in lines 5–17 leading to a count of 13. The differences in the counts are 18 to 17 to 13. One can easily see the potential for major discrepancies for large programs with many comments or programs written in languages that allow a large number of descriptive but non-executable statements. Conte [CONT86] has defined lines of code as:

"A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program header, declarations, and executable and non-executable statements".

This is the predominant definition for lines of code used by researchers. By this definition, Fig. 4.2 has 18 LOC.

There are some disadvantages of this simple method of counting. LOC is language dependent. A line of assembler is not the same as a line of COBOL. They also reflect what the system is rather than what it does. Measuring systems by the number of lines of code is rather like measuring a building by the number of bricks involved in construction; useful when deciding on the type and number of brick layers but useless in describing the building as a whole. Buildings are normally described in terms of facilities, the number and size of rooms, and their total areas in square feet or meters.

While lines of code have their uses (*e.g.*, in estimating programming time for a program during the build phase of a project), their usefulness is limited for other tasks like functionality, complexity, efficiency, etc. If counting LOC is similar to counting bricks in a building, then what is needed is some way of expressing the system in a way that is analogous to counting the number and size of rooms and their total area in square feet or meters.

4.1.2 Function Count

Measuring software size in terms of lines of code is analogous to measuring a car stereo by the number of resistors, capacitors and integrated circuits involved in its production. The number of components is useful in predicting the number of assembly line staff needed, but it does not say anything about the functions available in the finished stereo. When dealing with customers, the manufacturer talks in terms of functions available (*e.g.*, digital tuning) and not in terms of components (*e.g.*, integrated circuits).

Alan Albrecht while working for IBM, recognised the problem in size measurement in the 1970s, and developed a technique (which he called Function Point Analysis), which appeared to be a solution to the size measurement problem [ALBR79, ALBR83]. It measures functionality from the users point of view, that is, on the basis of what the user requests and receives in return. Therefore, it deals with the functionality being delivered, and not with the lines of code, source modules, files, etc. Measuring size in this way has the advantage that size measure is independent of the technology used to deliver the functions. In other words, two identical counting systems, one written in 4 GL and the other in assembler, would have the same function count. This makes sense to the user, because the object is to buy an accounting system, not lines of assembler and it makes sense to the IT department, because they can measure the performance differences between the assembler and 4GL environments [STEP95].

Function point measures functionality from the users point of view, that is, on the basis of what the user requests and receives in return from the system. The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units

- Inputs : information entering the system.
- Outputs : information leaving the system.
- Enquiries : requests for instant access to information.
- Internal logical files : information held within the system.
- External interface files : Information held by other systems that is used by the system being analyzed.

The FPA functional untis are shown in Fig. 4.3.

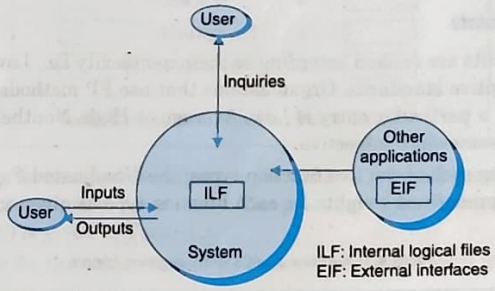

Fig. 4.3: FPAs functional units

The five functional units are divided in two categories:

(*i*) **Data function types**

- *Internal Logical files (ILF):* A user identifiable group of logically related data or control information maintained within the system.
- *External Interface files (EIF):* A user identifiable group of logically related data or control information referenced by the system, but maintained within another system. This means that EIF counted for one system, may be an ILF in another system.

(*ii*) **Transactional function types**

- *External Input (EI):* An EI processes data or control information that comes from outside the system. The EI is an elementary process, which is the smallest unit of activity that is meaningful to the end user in the business.
- *External Output (EO):* An EO is an elementary process that generates data or control information to be sent outside the system.
- *External Inquiry (EQ):* An EQ is an elementary process that is made up of an input-output combination that results in data retrieval.

Special features

- Function point approach is independent of the language, tools, or methodologies used for implementation; *i.e.*, they do not take into consideration programming languages, data base management systems, processing hardware or any other data base technology.
- Function points can be estimated from requirement specification or design specifications, thus making it possible to estimate development effort in early phases of development.
- Function points are directly linked to the statement of requirements; any change of requirements can easily be followed by a re-estimate [INCE89].
- Function points are based on the system user's external view of the system, non-technical users of the software system have a better understanding of what function points are measuring.

This method resolves many of the inconsistencies that arise when using lines of code as a software size measure [MATS94].

Counting function points

The five functional units are ranked according to their complexity *i.e.,* Low, Average, or High, using a set of prescriptive standards. Organisations that use FP methods develop criteria for determining whether a particular entry is Low, Average or High. Nontheless, the determination of complexity is somewhat subjective.

After classifying each of the five function types, the Unadjusted Function Points (UFP) are calculated using predefined weights for each function type as given in Table 4.1.

Table 4.1: Functional units with weighting factors

Functional units	*Weighting factors*		
	Low	*Average*	*High*
External Inputs (EI)	3	4	6
External Output (EO)	4	5	7
External Inquiries (EQ)	3	4	6
Internal logical files (ILF)	7	10	15
External Interface files (EIF)	5	7	10

Table 4.2: UFP calculation table

Functional units	*Count complexity*		*Complexity totals*	*Functional unit totals*
External Inputs (EIs)	☐	Low × 3 =	☐	☐
	☐	Average × 4 =	☐	
	☐	High × 6 =	☐	
External Outputs (EOs)	☐	Low × 4 =	☐	☐
	☐	Averge × 5 =	☐	
	☐	High × 7 =	☐	
External Inquiries (EQs)	☐	Low × 3 =	☐	☐
	☐	Average × 4 =	☐	
	☐	High × 6 =	☐	
Internal logical files (ILFs)	☐	Low × 7 =	☐	☐
	☐	Average × 10 =	☐	
	☐	High × 15 =	☐	
External Interface files (EIFs)	☐	Low × 5 =	☐	☐
	☐	Average × 7 =	☐	
	☐	High × 10 =	☐	
Total Unadjusted Function Point Count				☐

The weighting factors are identified (as per Table 4.1) for all functional units and multiplied with the functional units accordingly. The procedure for the calculation of Unadjusted Function Point (UFP) is given in Table 4.2.

The procedure for the calculation of UFP in mathematical form is given below:

$$\text{UFP} = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

where i indicates the row and j indicates the column of Table 4.1.

w_{ij} : It is the entry of the i^{th} row and j^{th} column of the Table 4.1.

Z_{ij} : It is the count of the number of functional units of Type i that have been classified as having the complexity corresponding to column j.

Organisations that use function point methods develop a criterion for determining whether a particular entry is Low, Average or High. Nonetheless, the determination of complexity is somewhat subjective.

The final number of function points is arrived at by multiplying the UFP by an adjustment factor that is determined by considering 14 aspects of processing complexity which are given in Table 4.3. This adjustment factor allows the UFP count to be modified by at most ± 35%. The final adjusted FP count is obtained by using the following relationship.

$$FP = UFP * CAF$$

Where CAF is complexity adjustment factor and is equal to $[0.65 + 0.01 \times \Sigma F_i]$. The F_i (i = 1 to 14) are the degrees of influence and are based on responses to questions noted in Table 4.3.

Table 4.3: Computing function points

Rate each factor on a scale of 0 to 5.
0 = No Influence; 1 = Incidental; 2 = Moderate; 3 = Average; 4 = Significant; 5 = Essential
Number of factors considered (F_i) .
1. Does the system require reliable backup and recovery?
2. Is data communication required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing heavily utilized operational environment?
6. Does the system require on line data entry?
7. Does the on line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on line?
9. Is the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organisations?
14. Is the application designed to facilitate change and ease of use by the user?

Uses of function points

The collection of function point data has two primary motivations. One is the desire by managers to monitor levels of productivity, for example, number of function points achieved per work hour expended. From this perspective, the manager is not concerned with when the function point counts are made, but only that the function points accurately describe the size of the final software project. In this instance, function points have an advantage over LOC in that they provide a more objective measure of software size by which to assess productivity.

Another use of function points is in the estimation of software development cost. There are only a few studies that address this issue, though it is arguably the most important potential use of function point data.

Functions points may compute the following important metrics:

- Productivity = FP/persons-months
- Quality = Defects/FP
- Cost = Rupees /FP
- Documentation = Pages of documentation per FP

These metrics are controversial and are not universally acceptable. There are standards issued by the International Function Point User Group (IFPUG, covering the Albrecht method) and the United Kingdom Function Point User Group (UFPGU, covering the MK11 method). An ISO standard for function point methods is also being developed.

The function point method continues to be refined. So if we intend to use it we should obtain copies of the latest international function point user group (IFPUG) guidelines and standards. IFPUG is the fastest growing non-profit software metrics user group in the world with an annul growth rate of 30%. Today it has grown to over 800 corporate members and over 1500 individual members in more than 50 countries.

Example 4.1

Consider a project with the following functional units:

Number of user inputs = 50
Number of user outputs = 40
Number of user enquiries = 35
Number of user files = 06
Number of external interfaces = 04

Assume all complexity adjustment factors and weighting factors are average.

Compute the function points for the project.

Solution

We know $$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij} w_{ij}$$

$$UFP = 50 \times 4 + 40 \times 5 + 35 \times 4 + 6 \times 10 + 4 \times 7$$
$$= 200 + 200 + 140 + 60 + 28 = 628$$
$$CAF = (0.65 + 0.01 \Sigma F_i)$$
$$= (0.65 + 0.01(14 \times 3)) = 0.65 + 0.42 = 1.07$$
$$FP = UFP \times CAF$$
$$= 628 \times 1.07 = 672.$$

Example 4.2

An application has the following:

10 low external inputs, 12 high external outputs, 20 low internal logical files, 15 high external interface files, 12 average external inequiries, and a value of complexity adjustment factor of 1.10.

What are the unadjusted and adjusted function point counts ?

Solution

Unadjusted function point counts may be calculated using as:

$$UFP = \sum_{i=1}^{5} \sum_{J=1}^{3} Z_{ij}\, w_{ij}$$

$$= 10 \times 3 + 12 \times 7 + 20 \times 7 + 15 + 10 + 12 \times 4$$

$$= 30 + 84 + 140 + 150 + 48$$

$$= 452$$

$$FP = UFP \times CAF$$

$$= 452 \times 1.10 = 497.2.$$

Example 4.3

Consider a project with the following parameters.

(*i*) External Inputs:
- (*a*) 10 with low complexity
- (*b*) 15 with average complexity
- (*c*) 17 with high complexity.

(*ii*) External Outputs:
- (*a*) 6 with low complexity
- (*b*) 13 with high complexity.

(*iii*) External Inquiries:
- (*a*) 3 with low complexity
- (*b*) 4 with average complexity
- (*c*) 2 with high complexity.

(*iv*) Internal logical files:
- (*a*) 2 with average complexity
- (*b*) 1 with high complexity.

(*v*) External Interface files:
- (*a*) 9 with low complexity.

In addition to above, system requires

(*i*) Significant data communication

(*ii*) Performance is very critical

(*iii*) Designed code may be moderately reusable

(*iv*) System is not designed for multiple installations in different organisations.

Other complexity adjustment factors are treated as average. Compute the function points for the project.

Solution

Unadjusted function points may be counted using Table 4.2.

Functional units	*Count*	*Complexity*			*Complexity totals*	*Functional unit totals*
External Inputs (EIs)	10	Low	× 3	=	30	
	15	Average	× 4	=	60	
	17	High	× 6	=	102	192
External Outputs (EOs)	6	Low	× 4	=	24	
	0	Average	× 5	=	0	
	13	High	× 7	=	91	115
External Inquiries	3	Low	× 3	=	9	
	4	Average	× 4	=	16	
(EIs)	2	High	× 6	=	12	37
Internal Logical Files (ILFs)	0	Low	× 7	=	0	
	2	Average	× 10	=	20	
	1	High	× 15	=	15	35
External Interface Files (EIFs)	9	Low	× 5	=	45	
	0	Avergae	× 7	=	0	
	0	High	× 10	=	0	45
		Total unadjusted function point count =				424

The factors given in Table 4.3 may be calculated as:

$$\sum_{i=1}^{14} F_i = 3 + 4 + 3 + 5 + 3 + 3 + 3 + 3 + 3 + 3 + 2 + 3 + 0 + 3 = 41$$

$$CAF = (0.65 + 0.01 \times \Sigma F_i)$$

$$= (0.65 + 0.01 \times 41)$$

$$= 1.06$$

$$FP = UFP \times CAF$$

$$= 424 \times 1.06$$

$$= 449.44$$

Hence **FP = 449**

4.2 COST ESTIMATION

For any new software project, it is necessary to know how much will it cost to develop and how much development time will it take. These estimates are needed before development is initiated. But how is this done? In many cases estimates are made using past experience as the only guide. However, in most of the cases projects are different and hence past experience alone may not be enough. A number of estimation techniques have been developed and are having following attributes in common.

- Project scope must be established in advance
- Software metrics are used as a basis from which estimates are made
- The project is broken into small pieces which are estimated individually

To achieve reliable cost and schedule estimates, a number of options arise:

- Delay estimation until late in project (obviously we can achieve 100% accurate estimates after project is complete!)
- Use simple decomposition techniques to generate project cost and schedule estimates
- Develop empirical models for estimation
- Acquire one or more automated estimation tools

Unfortunately, the first option, however, attractive, is not practical. Cost estimates must be provided up front. However, we should recognise that the longer we wait, more we know, and more we know, the less likely, are we to make serious errors in our estimates [PRES2K]

4.3 MODELS

The model is concerned with the representation of the process to be estimated. A model may be static or dynamic. In a static model, a unique variable (say, size) is taken as a key element for calculating all others (say, cost, time). The form of equation used is the same for all calculations. In a dynamic model, all variables are interdependent and there is no basic variable as in the static model.

When a model makes use of a single basic variable to calculate all others it is said to be a *single-variable* model. In some models, several variables are needed to describe the software development process, and selected equations combine these variables to give the estimate of time and cost. These models are called *multivariable*. The variables, single or multiple, that are input to the model to predict the behaviour of a software development are called *predictors*. The choice and handling of these predictors are most crucial activity in estimating methodology [LOND87].

4.3.1 Static, Single Variable Models

Methods using this model use an equation to estimate the desired values such as cost, time, effort, etc. They all depend on the same variable used as predictor (say, size). An example of the most common equation is

$$C = a\,L^b \tag{4.1}$$

where C is the cost (effort expressed in any unit of manpower, for example, person-months) and L is the size generally given in the number of lines of code. The constants, a & b are

derived from the historical data of the organisation. Since a and b depend on the local development environment, these models are not transportable to different organisations.

The Software Engineering Laboratory of the University of Maryland has established a model, the SEL model, for estimating its own software productions. This model [BASL80] is a typical example of a static single-variable model.

$$E = 1.4\,L^{0.93} \tag{4.2}$$

$$DOC = 30.4\,L^{0.90} \tag{4.3}$$

$$D = 4.6\,L^{0.26} \tag{4.4}$$

Effort (E in Person-months), documentation (DOC, in number of pages) and duration (D, in months) are calculated from the number of lines of code (L, in thousands of lines) used as a predictor.

4.3.2 Static, Multivariable Models

Although these models are often based on equation (4.1), they actually depend on several variables representing various aspects of the software development environment, for example, methods used, user participation, customer oriented changes, memory constraints, etc. The model developed by Walston and Felix at IBM [WALS77] provides a relationship between delivered lines of source code (L in thousands of lines) and effort E (E in person- months) and is given by the following equation:

$$E = 5.2\,L^{0.91} \tag{4.5}$$

In the same fashion, the duration of the development (D in months) in given by

$$D = 4.1\,L^{0.36} \tag{4.6}$$

Data collected on 60 software projects, representing a wide variety of applications and size (ranging from 4000 to 467000 lines of code), shows a relationship between productivity (expressed in number of lines of source code per person months) and a productivity index I.

The productivity index uses 29 variables which are found to be highly correlated to productivity as follows:

$$I = \sum_{i=1}^{29} W_i X_i \tag{4.7}$$

where W_i is a factor weight for the i^{th} variable and $X_i = \{-1, 0, +1\}$. The estimator gives X_i one of the values -1, 0 or +1 depending on whether the variable decreases, has no effect, or increases the productivity respectively. The terms of equation (4.7) are then added up to give the productivity index. A productivity range can be obtained for the project by using a productivity versus index chart [LOND87].

Example 4.4

Compare the Walston-Felix model [equation (4.5) and equation (4.6)] with the SEL model [equation (4.2) and equation (4.4)] on a software development expected to involve 8 person-years of effort [LOND87].

(*a*) Calculate the number of lines of source code that can be produced.

(*b*) Calculate the duration of the development.

(c) Calculate the productivity in LOC/PY.

(d) Calculate the average manning.

Solution

The amount of manpower involved = 8 PY = 96 person-months

(a) Number of lines of source code can be obtained by reversing equation (4.2) and equation (4.5) to give:

$$L = (E/a)^{1/b}$$

Then

$$L(SEL) = (96/1.4)^{1/0.93} = 94264 \text{ LOC}$$

$$L(W\text{-}F) = (96/5.2)^{1/0.91} = 24632 \text{ LOC}$$

(b) Duration in months can be calculated by means of equation (4.4) and equation (4.6)

$$D(SEL) = 4.6^{0.26} L$$

$$= 4.6(94.264)^{0.26} = 15 \text{ months}$$

$$D(W\text{-}F) = 4.1 L^{0.36}$$

$$= 4.1 (24.632)^{0.36} = 13 \text{ months}$$

(c) Productivity is the lines of code produced per person/month (year).

$$P(SEL) = \frac{94264}{8} = 11783 \text{ LOC/Person-Years}$$

$$P(W\text{-}F) = \frac{24632}{8} = 3079 \text{ LOC/Person-Years}$$

(d) Average manning is the average number of persons required per month in the project.

$$M(SEL) = \frac{96 \text{ P-M}}{15 \text{ M}} = 6.4 \text{ Persons}$$

$$M(W\text{-}F) = \frac{96 \text{ P-M}}{13 \text{ M}} = 7.4 \text{ Persons}$$

If we look at the value of "L", it seems that SEL can produce four times as much software as IBM for the same manpower and time scale.

4.4 THE CONSTRUCTIVE COST MODEL (COCOMO)

This model gained rapid popularity following the publication of B.W. Boehm's excellent book *Software Engineering Economics* in 1981 [BOEH81]. COCOMO is a hierarchy of software cost estimation models, which include basic, intermediate and detailed sub models.

4.4.1 Basic Model

The basic model aims at estimating, in a quick and rough fashion, most of the small to medium sized software projects. Three modes of software development are considered in this model: organic, semi-detached and embedded.

In the organic mode, a small team of experienced developers develops software in a very familiar environment. The size of the software development in this mode ranges from small

(a few KLOC) to medium (a few tens of KLOC), while in other two modes the size ranges from small to very large (a few hundreds of KLOC).

In the embedded mode of software development, the project has tight constraints, which might be related to the target processor and its interface with the associated hardware. The problem to be solved is unique and so it is often hard to find experienced persons, as the same does not usually exist.

The *semi detached* mode is an intermediate mode between the organic mode and embedded mode. The comparision of all three modes is given in Table 4.4.

Table 4.4: The comparison of three COCOMO modes

Mode	*Project size*	*Nature of project*	*Innovation*	*Deadline of the project*	*Development environment*
Organic	Typically 2 – 50 KLOC	Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc.	Little	Not tight	Familiar & In house
Semi detached	Typically 50-300 KLOC	Medium size project, Medium size team, Average previous experience on similar projects. For Example: Utility systems like compilers, database systems, editors etc.	Medium	Medium	Medium
Embedded	Typically over 300 KLOC	Large project, Real time systems, Complex interfaces, Very little previous experience. For Example: ATMs, Air Traffic Control etc.	Significant	Tight	Complex Hardware/ customer Interfaces required

Depending on the problem at hand, the team might include a mixture of experienced and less experienced people with only a recent history of working together. The basic COCOMO equations take the form

$$E = a_b(KLOC)^{b_b} \quad (4.8)$$

$$D = c_b(E)^{d_b} \quad (4.9)$$

where E is effort applied in Person-Months, and D is the development time in months. The coefficients a_b, b_b, c_b and d_b are given in Table 4.4(a).

Table 4.4(a): Basic COCOMO co-efficients

Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

When effort and development time are known, the average staff size to complete the project may be calculated as:

$$\text{Average staff size (SS)} = \frac{E}{D} \text{ Persons.}$$

When project size is known, the productivity level may be calculated as:

$$\text{Productivity (P)} = \frac{KLOC}{E} \text{ KLOC/PM.}$$

With the basic model, the software estimator has a useful tool for estimating quickly, by two runs on a pocket calculator, the cost and development time of a software project, once the size is estimated. The software estimator will have to assess by himself/herself which mode is the most appropriate.

Example 4.5

Suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three modes *i.e.*, organic, semidetached and embedded.

Solution

The basic COCOMO equations take the form:

$$E = a_b(KLOC)^{b_b}$$

$$D = c_b(KLOC)^{d_b}$$

Estimated size of the project = 400 KLOC

(*i*) **Organic mode**

$$E = 2.4(400)^{1.05} = 1295.31 \text{ PM}$$

$$D = 2.5(1295.31)^{0.38} = 38.07 \text{ M}$$

(*ii*) **Semidetached mode**

$$E = 3.0(400)^{1.12} = 2462.79 \text{ PM}$$

$$D = 2.5(2462.79)^{.35} = 38.45 \text{ M}$$

(*iii*) **Embedded mode**

$$E = 3.6(400)^{1.20} = 4772.81 \text{ PM}$$

$$D = 2.5(4772.8)^{.32} = 38 \text{ M}$$

As we have seen, effort calculated for embedded mode is approximately 4 times, the effort for organic mode. However, the effort calculated for semidetached mode is 2 times the effort of organic mode. There is a large difference in these values. But, surprisingly, the development time is approximately the same for all three modes. It is clear from here that the

selection of mode is very important. Since, development time is approximately the same, the only varying parameter is the requirement of persons. Every mode will have different manpower requirement. If we look to the Table 4.4, it is mentioned that for over 300 KLOC projects, embedded mode is the right choice. The selection of a mode is not only dependent on project size, but also on other parameters as mentioned in Table 4.4. We should be utmost careful about the selection of mode for the project.

Example 4.6

A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the effort, development time, average staff size and productivity of the project.

Solution

The semi-detached mode is the most appropriate mode; keeping in view the size, schedules and experience of the development team.

Hence $E = 3.0(200)^{1.12} = 1133.12 \text{ PM}$

$D = 2.5(1133.12)^{.35} = 29.3 \text{ M}$

$$\text{Average staff size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons.}$$

$$\text{Productivity} = \frac{KLOC}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

$P = 176$ LOC/PM.

4.4.2 Intermediate Model

The basic model allowed for a quick and rough estimate, but it resulted in a lack of accuracy. Boehm introduced an additional set of 15 predictors called cost drivers in the intermediate model to take account of the software development environment. Cost drivers are used to adjust the nominal cost of a project to the actual project environment, hence increasing the accuracy of the estimate.

The cost drivers are grouped into four categories:

1. Product attributes
   - (*a*) Required software reliability (RELY)
   - (*b*) Database size (DATA)
   - (*c*) Product complexity (CPLX)
2. Computer attributes
   - (*a*) Execution time constraint (TIME)
   - (*b*) Main storage constraint (STOR)
   - (*c*) Virtual machine volatility (VIRT)
   - (*d*) Computer turnaround time (TURN)

3. Personnel attributes
   (a) Analyst capability (ACAP)
   (b) Application experience (AEXP)
   (c) Programmer capability (PCAP)
   (d) Virtual machine experience (VEXP)
   (e) Programming language experience (LEXP)
4. Project attributes
   (a) Modern programming practices (MODP)
   (b) Use of software tools (TOOL)
   (c) Required development schedule (SCED)

Each cost driver is rated for a given project environment. The rating uses a scale very low, low, nominal, high, very high, extra high which describes to what extent the cost driver applies to the project being estimated. Table 4.5 gives the multiplier values for the 15 cost drivers and their rating as provided by Boehm [BOEH81].

Table 4.5: Multiplier values for effort calculations

Cost Drivers	RATINGS					
	Very low	*Low*	*Nominal*	*High*	*Very high*	*Extra high*
Product attributes						
RELY	0.75	0.88	1.00	1.15	1.40	–
DATA	–	0.94	1.00	1.08	1.16	–
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer attributes						
TIME	–	–	1.00	1.11	1.30	1.66
STOR	–	–	1.00	1.06	1.21	1.56
VIRT	–	0.87	1.00	1.15	1.30	–
TURN	–	0.87	1.00	1.07	1.15	–
Personnel attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	–
AEXP	1.29	1.13	1.00	0.91	0.82	–
PCAP	1.42	1.17	1.00	0.86	0.70	–
VEXP	1.21	1.10	1.00	0.90	–	–
LEXP	1.14	1.07	1.00	0.95	–	–
Project attributes						
MODP	1.24	1.10	1.00	0.91	0.82	–
TOOL	1.24	1.10	1.00	0.91	0.83	–
SCED	1.23	1.08	1.00	1.04	1.10	–

The multiplying factors for all 15 cost drivers are multiplied to get the effort adjustment factor (EAF). Typical values for EAF range from 0.9 to 1.4.

The intermediate COCOMO equations take the form:

$$E = a_i(KLOC)^{b_i} * EAF \quad (4.10)$$

$$D = c_i(E)^{d_i} \quad (4.11)$$

The co-efficients a_i, b_i, c_i and d_i are given in Table 4.6.

Table 4.6: Co-efficients for intermediate COCOMO

Project	a_i	b_i	c_i	d_i
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

4.4.3 Detailed COCOMO Model

A large amount of work has been done by Boehm to capture all significant aspects of a software development. It offers a means for processing all the project characteristics to construct a software estimate. The detailed model introduces two more capabilities:

1. Phase-sensitive effort multipliers:
   Some phases (design, programming, integration/test) are more affected than others by factors defined by the cost drivers. The detailed model provides a set of phase sensitive effort multipliers for each cost driver. This helps in determining the manpower allocation for each phase of the project.
2. Three-level product hierarchy:
   Three product levels are defined. These are module, subsystem and system levels. The ratings of the cost drivers are done at appropriate level; that is, the level at which it is most susceptible to variation.

Development phases

A software development is carried out in four successive phases: plans/requirements, product design, programming and integration/test.

1. **Plan/requirements:** This is the first phase of the development cycle. The requirement is analyzed, the product plan is set up and a full product specification is generated. This phase consumes from 6% to 8% of the effort and 10% to 40% of the development time. These percentages depend not only on mode (organic, semi-detached or embedded), but also on the size.
2. **Product design:** The second phase of the COCOMO development cycle is concerned with the determination of the product architecture and the specification of the subsystem. This phase requires from 16% to 18% the nominal effort and can last from 19% to 38% of the development time.

3. **Programming:** The third phase of the COCOMO development cycle is divided into two sub phases: detailed design and code/unit test. This phase requires from 48% to 68% of the effort and lasts from 24% to 64% of the development time.
4. **Integration/Test:** This phase of the COCOMO development cycle occurs before delivery. This mainly consists of putting the tested parts together and then testing the final product. This phase requires from 16% to 34% of the nominal effort and can last from 18% to 34% of the development time.

Principle of the effort estimate

Size equivalent: As the software might be partly developed from software already existing (that is, re-usable code), a full development is not always required. In such cases, the parts of design document (DD%), code (C%) and integration (I%) to be modified are estimated. Then, an adjustment factor, A, is calculated by means of the following equation.

$$A = 0.4\ DD + 0.3\ C + 0.3\ I \tag{4.12}$$

The size equivalent is obtained by

$$S(\text{equivalent}) = (S \times A)/100 \tag{4.13}$$

where S represents the thousands of lines of code (KLOC) of the module. Multipliers have been developed that can be applied to the total project effort, E, and total project development time, D in order to allocate effort and schedule components to each phase in the life cycle of a software development program. There are assumed to be five distinct life cycle phases, and the effort and schedule for each phase are assumed to be given in terms of the overall effort and schedule by

$$E_p = \mu_p E \tag{4.14}$$

$$D_p = \tau_p D \tag{4.15}$$

where μ_p and τ_p are given in Table 4.7. There exist more sophisticated versions of this development that result in multipliers μ_p and τ_p that not only depend on the particular phase of the life cycle and mode of operation of the software but also contain the correction terms for the 15 attributes [BOEH81].

The COCOMO model is certainly the most thoroughly documented model currently available. It is very easy to use. And by doing so, the software manager can learn a lot about productivity, particularly from the very clear presentation of the cost drivers. The size and cost drivers can be progressively adjusted to realistic values to some extent. However, mode choice offers some difficulties, since it is not always possible to be sure which of the three modes is appropriate for a given software development—it might be a mixed mode.

Table 4.7: Effort and schedule fractions occurring in each phase of the lifecycle [SAGE90]

Mode & code size	*Plan & require-ment*	*System design*	*Detail design*	*Module code & test*	*Integration and test*
Lifecycle Phase value of μ_p					
Organic Small S=2	0.06	0.16	0.26	0.42	0.16
Organic Medium S=32	0.06	0.16	0.24	0.38	0.22
Semidetached Medium S=32	0.07	0.17	0.25	0.33	0.25
Semidetached Large S=128	0.07	0.17	0.24	0.31	0.28
Embedded Large S=128	0.08	0.18	0.25	0.26	0.31
Embedded Extra Large S=320	0.08	0.18	0.24	0.24	0.34
Lifecycle Phase value of τ_p					
Organic Small S=2	0.10	0.19	0.24	0.39	0.18
Organic Medium S=32	0.12	0.19	0.21	0.34	0.26
Semidetached Medium S=32	0.20	0.26	0.21	0.27	0.26
Semidetached Large S=128	0.22	0.27	0.19	0.25	0.29
Embedded Large S=128	0.36	0.36	0.18	0.18	0.28
Embedded Extra Large S=320	0.40	0.38	0.16	0.16	0.30

There are five phases of software life cycle in Table 4.7. However, plan and requirement phase has been combined with system design and known as requirement and product design. Both include the most conceptual part of the life cycle. So the effort and time shown in Table 4.7 for plan and requirements phase are over and above the estimated effort and time. The actual distribution may start from system design phase. Hence, four phases are:

1. Requirement and product design
   (*a*) Plans and requirements
   (*b*) System design
2. Detailed Design
   (*a*) Detailed design
3. Code & Unit test
   (*a*) Module code & test
4. Integrate and Test
   (*a*) Integrate & Test.

COCOMO is highly calibrated model, based on previous experience. It is easy to use and documented properly. Actual data gathered from previous projects may help to determine the values of the constants of the model (like *a*, *b*, *c* and *d*). These values may very from organisation to organisation. However, this model ignores software safety & security issues. It also ignores many hardware and customer related issues. It is silent about the involvement and responsiveness of customer.

It does not give proper importance to software requirements and specification phase which has identified as the most sensitive phase of software development life cycle.

Example 4.7

A new project with estimated 400 KLOC embedded system has to be developed. Project manager has a choice of hiring from two pools of developers: Very highly capable with very little experience in the programming language being used or developers of low quality but a lot of experience with the programming language. What is the impact of hiring all developers from one or the other pool ?

Solution

This is the case of embedded mode and model is intermediate COCOMO.

Hence $E = a_i(KLOC)^{d_i}$

$= 2.8(400)^{1.20} = 3712$ PM

Case I: Developers are very highly capable with very little experience in the programming being used.

$EAF = 0.82 \times 1.14 = 0.9348$

$E = 3712 \times .9348 = 3470$ PM

$D = 2.5(3470)^{0.32} = 33.9$ M

Case II: Developers are of low quality but lot of experience with the programming language being used.

$EAF = 1.29 \times 0.95 = 1.22$

$E = 3712 \times 1.22 = 4528$ PM

$D = 2.5(4528)^{0.32} = 36.9$ M

Case II requires more effort and time. Hence, low quality developers with lot of programming language experience could not match with the performance of very highly capable developers with very little experience.

Example 4.8

Consider a project to develop a full screen editor. The major components identified are (1) Screen Edit (2) Command language Interpreter (3) File input and output, (4) Cursor movement and (5) Screen movement. The sizes for these are estimated to be 4K, 2K, 1K, 2K and 3K delivered source code lines. Use COCOMO model to determine:

(*a*) Overall cost and schedule estimates (assume values for different cost drivers, with at least three of them being different from 1.0).

(*b*) Cost and Schedule estimates for different phases.

Solution

Size of five modules are:

Screen edit	= 4 KLOC
Command language interpreter	= 2 KLOC
File input and output	= 1 KLOC
Cursor movement	= 2 KLOC
Screen movement	= 3 KLOC
Total	**= 12 KLOC**

Let us assume that significant cost drivers are

(*i*) Required software reliability is high, *i.e.*, 1.15

(*ii*) Product complexity is high, *i.e.*, 1.15

(*iii*) Analyst capability is high, *i.e.*, 0.86

(*iv*) Programming language experience is low, *i.e.*, 1.07

(*v*) All other drivers are nominal.

$EAF = 1.15 \times 1.15 \times 0.86 \times 1.07 = 1.2169$

(*a*) The initial effort estimate for the project is obtained from the following equation

$E = a_i(KLOC)^{b_i} \times EAF$

$= 3.2(12)^{1.05} \times 1.2169 = 52.91$ PM

Development time $D = C_i(E)^{d_i}$

$= 2.5\ (52.91)^{0.38} = 11.29$ M

(*b*) Using the following equations and referring Table 4.7, phase wise cost and schedule estimates can be calculated.

$E_p = \mu_p E$

$D_p = \tau_p D$

Since size is only 12 KLOC, it is an organic small model. Phase wise effort distribution is given below:

System Design	$= 0.16 \times 52.91 = 8.465$ PM
Detailed Design	$= 0.26 \times 52.91 = 13.756$ PM
Module Code & Test	$= 0.42 \times 52.91 = 22.222$ PM
Integration & Test	$= 0.16 \times 52.91 = 8.465$ PM

Now Phase wise development time duration is

System Design	$= 0.19 \times 11.29 = 2.145$ M
Detailed Design	$= 0.24 \times 11.29 = 2.709$ M
Module Code & Test	$= 0.39 \times 11.29 = 4.403$ M
Integration & Test	$= 0.18 \times 11.29 = 2.032$ M

4.5 COCOMO-II

COCOMO-II is the revised version of the original COCOMO (discussed in article 4.4) and is developed at University of Southern California under the leadership of Dr. Barry Boehm. The model is tuned to the life cycle practices of the 21st century. It also provides a quantitative analytic framework, and set of tools and techniques for evaluating the effects of software technology improvements on software life cycle costs and schedules. The following categories of applications/projects are identified by COCOMO-II for the estimation [UCSD01] and are shown in Fig. 4.4.

(*i*) **End user programming:** This category is applicable to small systems, developed by end user using application generators. Some application generators are spreadsheets, extended query system, report generators etc. End user may write small programs using application generators. The end users may not have sufficient knowledge about computers and software engineering practices. However, they may have in-depth knowledge about their business needs and practices. Hence, this excellent domain knowledge may motivate them to develop an application using user friendly tools like MS-Excel, MS-Access, MS-Studio etc.

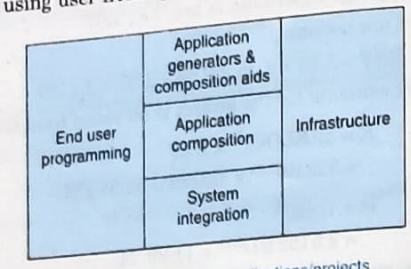

Fig. 4.4: Categories of applications/projects

(*ii*) **Infrastructure sector:** This category is applicable to the infrastructure development *i.e.*, the software that provides infrastructure like operating systems, database management systems, user interface management system, networking system etc. Some commercial examples are Microsoft products, Oracle, DB2, MAYA, LINUX, 3D-STUDIO.

Infrastructure developers generally have good knowledge of software development and software engineering practices and relatively little knowledge about applications. The product lines will have many reusable components, but the pace of technology (new processor, memory, communications, display, and multimedia technology) will require them to build many components and capabilities from scratch.

(*iii*) **Intermediate sectors:** This category is partitioned in three sub categories as shown in Fig. 4.4. Software developers will need to have good knowledge of software development and software engineering practices, experience and expertise of infrastructure software and indepth domain knowledge of one or more applications. Creating this talent pool is a major challenge.

Application generators and composition aids : This subcategory will create largely prepackaged capabilities for user programming. Typical firms operating in this sector are Microsoft, Lotus, Novell, Borland, Alias Wavefront, Oracle, IBM. Their product lines will have many reusable components, but also will require a good deal of new capability development from the scratch. Application composition aids will be developed both by the firms above and by software product line investments of firms in the application composition sector.

Application composition sector : This subcategory deals with applications which are too diversified to be handled by prepackaged solutions, but which are sufficiently simple to be rapidly composable from interoperable components. Typical components will be graphic user interface (GUI) builders, databases or object managers, domain specific components such as financial, medical, or industrial process control packages etc.

These applications are complex, large, versatile, diversified and require specialised developers with sound knowledge of development and software engineering practices.

However, they are developed using application generator environment like CASE tools, DBMS (DB2, oracle etc.), and 4GL programming tools (Developer 2000, Visual basic, Power builder, ASP, JSP, PHP etc).

System integration : This subcategory deals with large scale, highly embedded, or unprecedented systems. Portions of these systems can be developed with application composition capabilities, but their demands generally require a significant amount of upfront systems engineering and customised software development activities.

Stages of COCOMO-II : The end user programming sector does not need a COCOMO-II model. Its applications are normally developed in hours to days. Hence a simple activity based estimate will generally be sufficient.

COCOMO-II includes three stages. Stage I supports estimation of prototyping or application composition types of projects. Stage II supports estimation in the early design stage of a project, when less is known about the project's cost drivers. Stage III supports estimation in the Post-Architecture stage of a project. The details are given in Table 4.8.

Table 4.8: Stages of COCOMO-II

Stage No.	*Model name*	*Applicable for the types of projects*	*Applications*
Stage I	Application composition estimation model	Application composition	In addition to application composition type of projects, this model is also used for prototyping (if any) stage of application generators, infrastructure & system integration.
Stage II	Early design estimation model	Application generators, infrastructure & system integration.	Used in early design stage of a project, when less is known about the project.
State III	Post architecture estimation model	Application generators, infrastructure & system integration	Used after the completion of the detailed architecture of the project

4.5.1 Application Composition Estimation Model

The model is designed for quickly developed applications using interoperable components. Example of these components based systems are Graphic User Interface (GUI) builders, database or object managers, hypermedia handlers, smart data finders, and domain specific components such as financial, medical, or industrial process control packages. The model can also be used for the prototyping phase of application generator development, infrastructure sector and system integration projects.

In this model, size is first estimated using object points. The object points are easy to identify and count. The object in object points defines screens, reports, and 3GL modules as objects. This may or may not have any relationship to other definitions of "objects", such as those processing features like class affiliation, inheritance, encapsulation, message passing, and so forth [UCSDOI].

Object point estimation is a relatively new size estimation technique, but it is well suited in application composition sector. It is also a good match to associated prototyping efforts, based on the use of a rapid composition Integrated Computer Aided Software Engineering (ICASE) Environment providing graphic user interface builders, software development tools, and large, composable infrastructure and applications components. The steps required for the estimation of effort in Person-months are given in Fig. 4.5.

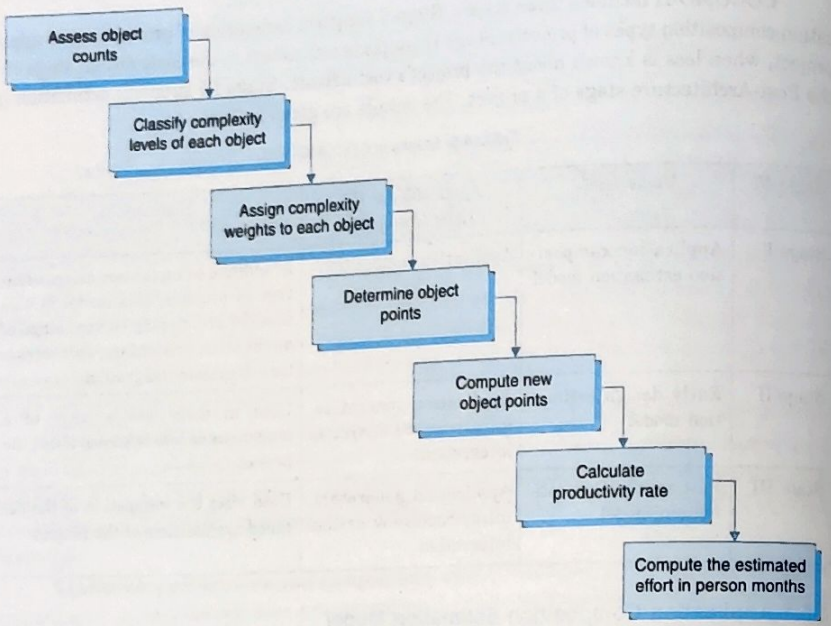

Fig. 4.5: Steps for the estimation of effort in person months

(*i*) **Assess object counts:** Estimate the number of screens, reports, and 3GL components that will comprise this application.

(*ii*) **Classification of complexity levels:** We have to classify each object instance into simple, medium and difficult complexity levels depending on values of its characteristics.

The screens are classified on the basis of number of views and sources and reports are on the basis of number of sections and sources. The details are given in Table 4.9.

Table 4.9(a): For screens

Number of views contained	*# and sources of data tables*		
	Total < 4 (< 2 server < 3 client)	*Total < 8 (2 – 3 server 3 – 5 client)*	*Total 8 + (> 3 server, > 5 client)*
< 3	Simple	Simple	Medium
3 – 7	Simple	Medium	Difficult
> 8	Medium	Difficult	Difficult

Table 4.9(b): For reports

Number of sections contained	*# and sources of data tables*		
	Total < 4 (< 2 server < 3 client)	*Total < 8 (2 – 3 server 3 – 5 client)*	*Total 8 + (> 3 server, > 5 client)*
0 or 1	Simple	Simple	Medium
2 or 3	Simple	Medium	Difficult
4 +	Medium	Difficult	Difficult

(*iii*) **Assign comlexity weight to each object:** The weights are used for three object types *i.e.,* screen, report and 3GL components using the Table 4.10.

The weights reflect the relative effort required to implement an instance of that complexity level.

Table 4.10: Complexity weights for each level

Object type	*Complexity weight*		
	Simple	*Medium*	*Difficult*
Screen	1	2	3
Report	2	5	8
3GL Component	—	—	10

(*iv*) **Determine object points:** Add all the weighted object instances to get one number and this number is known as object-point count.

(*v*) **Compute new object points:** We have to estimate the percentage of reuse to be achieved in a project. Depending on the percentage reuse, the new object points (NOP) are computed.

$$\text{NOP} = \frac{(\text{object points}) * (100 - \%\ \text{reuse})}{100}$$

NOPs are the object points that will need to be developed and differ from the object point count because there may be reuse.

(*vi*) **Calculation of productivity rate:** The productivity rate can be calculated as:

Productivity rate (PROD) = NOP/Person month.

PROD values Calculation requires NOP and total person-months of past projects in similar environments. COCOMO-II application composition model gives the following Table 4.11 containing the values of PROD based on their data and experience.

Table 4.11: Productivity values

Developer's experience & capability; ICASE maturity & capability	*PROD (NOP/PM)*
Very low	4
Low	7
Nominal	13
High	25
Very high	50

(*vii*) **Compute the effort in person-months:** When PROD is known, we may estimate effort in Person-Months as:

$$\text{Effort in PM} = \frac{\text{NOP}}{\text{PROD}}$$

Example 4.9

Consider a database application project with the following characteristics:

(*i*) The application has 4 screens with 4 views each and 7 data tables for 3 servers and 4 clients.

(*ii*) The application may generate two report of 6 sections each from 07 data tables for two server and 3 clients. There is 10% reuse of object points.

The developer's experience and capability in the similar environment is low. The maturity of organisation in terms of capability is also low. Calculate the object point count, New object points and effort to develop such a project.

Solution

This project comes under the category of application composition estimation model.

Number of screens = 4 with 4 views each

Number of reports = 2 with 6 sections each.

From Table 4.9, we know that each screen will be of medium complexity and each report will be of difficult complexity.

Using Table 4.10 of complexity weights, we may calculate object point count

$$= 4 \times 2 + 2 \times 8 = 24$$

$$\text{NOP} = \frac{24 * (100 - 10)}{100} = 21.6$$

Table 4.11 gives the low value of productivity (PROD) *i.e.*, 7.

$$\text{Efforts in PM} = \frac{\text{NOP}}{\text{PROD}}$$

$$\text{Effort} = \frac{21.6}{7} = 3.086 \text{ PM}$$

4.5.2 The Early Design Model

The COCOMO-II models use the base equation of the form

$$\text{PM}_{\text{nominal}} = \text{A} * (\text{size})^{\text{B}}$$

where $\text{PM}_{\text{nominal}}$ = Effort of the project in person months.

A = Constant representing the nominal productivity, provisionally set to 2.5

B = Scale factor

Size = Software size

The early design model uses Unadjusted Function Points (UFP) as the measure of size. This model is used in the early stages of a software project when very little may be known about the size of the product to be developed, the nature of the target platform, the nature of the personnel to be involved in the project or the detailed specifics of the process to be used. This model can be used in either Application Generator, System Integration, or Infrastructure Development Sector.

If B = 1.0, there is a linear relationship of effort and size. If value of B is not 1, there will be a non-linear relationship between size and effort. If B < 1.0, the rate of increase of effort decreases as the size of the product increases. If the product's size is doubled, the project effort is less than doubled.

If B > 1.0, the rate of increase of effort increases as the size of the product increases. This is due to the growth of interpersonal communications overheads and growth of large system integration overhead. Application composition model assumes the value of B to be 1. But the early design model assumes the value of B to be greater than 1. Thus, the basic assumption is that the effort spent in a project usually increases faster than the size of the project. The value of B is computed on the basis of scaling factors (or drivers) that may cause drop in productivity with increase in size.

Table 4.12: Scaling factors required for the calculation of the value of B

Scale factor	*Explanation*	*Remarks*
Precedentness	Reflects the previous experience on similar projects. This is applicable to individuals & organisation both in terms of expertise & experience.	Very low means no previous experiences, Extra high means that organisation is completely familiar with this application domain.
Development flexibility	Reflects the degree of flexibility in the development process.	Very low means a well defined process is used. Extra high means that the client gives only general goals.
Architecture/Risk resolution	Reflects the degree of risk analysis carried out.	Very low means very little analysis and Extra high means complete and thorough risk analysis.

(Contd...)

Team cohesion	Reflects the team management skills.	Very low means very little interaction & hardly any relationship among team members; Extra high means an integrated & effective team.
Process maturity	Reflects the process maturity of the organisation. Thus it is dependent on SEI-CMM level of the organisation.	Very low means organisation has no level at all and extra high means organisation is rated as highest level of SEI-CMM.

The scaling factors that COCOMO-II uses for the calculation of B are Precedentness, Development Flexibility, Architecture/Risk Resolution, Team Cohesion and Process Maturity. The details are given in Table 4.12. These factors are rated on a six point scale *i.e.*, very low, low, nominal, high, very high and extra high and are given in Table 4.13.

Table 4.13: Data for the Computation of B

Scaling factors	*Very low*	*Low*	*Nominal*	*High*	*Very high*	*Extra high*
Precedentness	6.20	4.96	3.72	2.48	1.24	0.00
Development flexibility	5.07	4.05	3.04	2.03	1.01	0.00
Architecture/Risk resolution	7.07	5.65	4.24	2.83	1.41	0.00
Team cohesion	5.48	4.38	3.29	2.19	1.10	0.00
Process maturity	7.80	6.24	4.68	3.12	1.56	0.00

The value of B can be calculated as:

$$B = 0.91 + 0.01 * (\text{Sum of rating on scaling factors for the project}).$$

When all the scaling factors of a project are rated as extra high, the best value of B is obtained and is equal to 0.91. When all the scaling factors are very low, the worst value of B is obtained and is equal to 1.23. Hence value of B may vary from 0.91 to 1.23.

Early design cost drivers

There are seven early design cost drivers and are given below:

(*i*) Product Reliability and Complexity (RCPX)
(*ii*) Required Reuse (RUSE)
(*iii*) Platform Difficulty (PDIF)
(*iv*) Personnel Capability (PERS)
(*v*) Personnel Experience (PREX)
(*vi*) Facilities (FCIL)
(*vii*) Schedule (SCED)

Post architecture cost drivers

There are 17 cost drivers in the Post Architecture model. These are rated on a scale of 1 to 6 as given below:

Very Low	*Low*	*Nominal*	*High*	*Very High*	*Extra High*
1	2	3	4	5	6

The list of seventeen cost drivers is given below:

1. Reliability Required (RELY)
2. Database Size (DATA)
3. Product Complexity (CPLX)
4. Required Reusability (RUSE)
5. Documentation (DOCU)
6. Execution Time Constraint (TIME)
7. Main Storage Constraint (STOR)
8. Platform Volatility (PVOL)
9. Analyst Capability (ACAP)
10. Programmers Capability (PCAP)
11. Personnel Continuity (PCON)
12. Analyst Experience (AEXP)
13. Programmer Experience (PEXP)
14. Language & Tool Experience (LTEX)
15. Use of Software Tools (TOOL)
16. Site Locations & Communication Technology between Sites (SITE)
17. Schedule (SCED)

Mapping of early design cost drivers and post architecture cost drivers

The 17 Post Architecture Cost Drivers are mapped to 7 Early Design Cost Drivers and are given in Table 4.14. This mapping is essential because many parameters will not be known correctly in early design phase. The mapping combines estimated parameters in order to have reasonable view of cost drivers. In Post Architecture, all 17 drivers will be known with reasonable accuracy, hence no mapping is required.

Table 4.14: Mapping table

Early design cost drivers	counter part combined post architecture cost drivers
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PERS	ACAP, PCAP, PCON
PREX	AEXP, PEXP, LTEX
FCIL	TOOL, SITE
SCED	SCED

Product of cost drivers for early design model

The combined early design cost drivers may be obtained by summing the numerical values of the contributing Post Architecture cost drivers. The resulting totals are allocated to an expanded early design model rating scale from Extra Low to Extra High. The early design model rating scales always have a Nominal total equal to the sum of the Nominal ratings of its contributing Post-Architecture Cost drivers.

(*i*) **Product reliability and complexity (RCPX):** The cost driver combines four Post-Architecture cost drivers which are RELY, DATA, CPLX and DOCU. Here RELY & DOCU range from Very Low to Very High. DATA ranges from Low to Very High; and CPLX ranges from Very Low to Extra High. The numerical sum of their ratings thus ranges from 5(VL, L, VL, VL) to 21(VH, VH, EH, VH). For details please refer to Table 4.16. The RCPX rating levels are given below:

RCPX	Extra low	Very low	Low	Nominal	High	Very high	Extra high
Sum of RELY, DATA, CPLX, DOCU ratings	5, 6	7, 8	9-11	12	13-15	16-18	19-21
Emphasis on reliability, documentation	Very Little	Little	Some	Basic	Strong	Very Strong	Extreme
Product complexity	Very Simple	Simple	Some	Moderate	Complex	Very Complex	Extremely Complex
Database size	Small	Small	Small	Moderate	Large	Very Large	Very Large

(*ii*) **Required reuse (RUSE):** This early design model cost driver is same as its Post-architecture Counterpart. The RUSE rating levels are (As per Table 4.16):

	Very low	Low	Nominal	High	Very high	Extra high
RUSE	1	2	3	4	5	6
		None	Across project	Across program	Across product line	Across multiple product line

(*iii*) **Platform difficulty (PDIF):** This cost driver combines TIME, STOR, and PVOL of Post-Architecture cost drivers. From the Table 4.16 it is clear that TIME and STOR range from Nominal to Extra High; PVOL ranges from Low to Very High. The numerical sum of ratings thus ranges from 8(N, N, L) to 17 (EH, EH, VH). Hence PDIF rating levels are:

PDIF	Low	Nominal	High	Very high	Extra high
Sum of Time, STOR & PVOL ratings	8	9	10-12	13-15	16-17
Time & storage constraint	≤ 50%	≤ 50%	65%	80%	90%
Platform Volatility	Very stable	Stable	Somewhat stable	Volatile	Highly volatile

(*iv*) **Personnel capability (PERS):** This cost driver combines three Post-Architecture Cost drivers. These drivers are analyst capability (ACAP), Programmers Capability (PCAP) and Personnel Continuity (PCON). Each of these has a rating scale from Very Low to very High as per Table 4.16. Adding up their numerical ratings produces values ranging from 3 to 15. PERS rating levels are calculated with the help of Table 4.16 and are given below:

PERS	Extra low	Very low	Low	Nominal	High	Very high	Extra high
Sum of ACAP, PCAP, PCON ratings	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Combined ACAP & PCAP Percentile	20%	39%	45%	55%	65%	75%	85%
Annual Personnel Turnover	45%	30%	20%	12%	9%	5%	4%

(*v*) **Personnel experience (PREX):** This early design cost driver combines three Post Architecture Cost drivers, which are: application experience (AEXP), platform experience (PEXP) and language and Tool experience (LTEX). Each of these range from Very Low to Very High and numerical sum of their ratings ranges from 3 to 15. The PREX rating levels are obtained using Table 4.16 and are given below:

PREX	Extra low	Very low	Low	Nominal	High	Very high	Extra high
Sum of AEXP, PEXP and LTEX ratings	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
Applications, Platform, Language & Tool Experience	≤ 3 months	5 months	9 months	1 year	2 year	4 year	6 year

(vi) **Facilities (FCIL):** This depends on two Post Architecture Cost drivers: Use of Software Tools (TOOL) and multisite development (SITE). TOOL ranges from Very Low to Very High; SITE ranges from Very Low to Extra High. Thus the numeric sum of their rating ranges from 2(VL, VL) to 11(VH, EH). FCIL rating levels are obtained using Table 4.16.

FCIL	Extra low	Very low	Low	Nominal	High	Very high	Extra high
Sum of TOOL & SITE ratings	2	3	4, 5	6	7, 8	9, 10	11
Tool support	Minimal	Some	Simple CASE tools	Basic life cycle tools	Good support of tools	Very strong use of tools	Very strong & well integrated tools
Multisite conditions development support	Weak support of complex multisite development	Some support	Moderate support	Basic support	Strong support	Very strong support	Very strong support

(vii) **Schedule (SCED):** The early design cost driver is the same as Post Architecture Counter part and rating levels are given below using Table 4.16.

SCED	Very Low	Low	Nominal	High	Very High
Schedule	75% of Nominal	85%	100%	130%	160%

The seven early design cost drivers have been converted into numeric values with a Nominal value 1.0. These values are used for the calculation of a factor called "Effort multiplier" which is the product of all seven early design cost drivers. The numeric values are given in Table 4.15.

Table 4.15: Early design Parameters

Early design cost drivers	Extra low	Very low	Low	Nominal	High	Very high	Extra high
RCPX	.73	.81	.98	1.0	1.30	1.74	2.38
RUSE	—	—	0.95	1.0	1.07	1.15	1.24
PDIF	—	—	0.87	1.0	1.29	1.81	2.61
PERS	2.12	1.62	1.26	1.0	0.83	0.63	0.50
PREX	1.59	1.33	1.12	1.0	0.87	0.71	0.62
FCIL	1.43	1.30	1.10	1.0	0.87	0.73	0.62
SCED	—	1.43	1.14	1.0	1.0	1.0	—

The early design model adjusts the nominal effort using 7 effort multipliers (EMs). Each effort multiplier (also called cost drivers) has 7 possible weights as given in Table 4.15. These factors are used for the calculation of adjusted effort as given below:

$$PM_{adjusted} = PM_{nominal} \times \left[\prod_{i=1}^{7} EM_i\right]$$

$PM_{adjusted}$ effort may very even up to 400% from $PM_{nominal}$.

Hence $PM_{adjusted}$ is the fine tuned value of effort in the early design phase.

Example 4.10

A software project of application generator category with estimated 50 KLOC has to be developed. The scale factor (B) has low precedentness, high development flexibility and low team cohesion. Other factors are nominal. The early design cost drivers like platform difficult (PDIF) and Personnel Capability (PERS) are high and others are nominal. Calculate the effort in person months for the development of the project.

Solution

Here $B = 0.91 + 0.01 * (\text{Sum of rating on scaling factors for the project})$

$= 0.91 + 0.01 * (4.96 + 2.03 + 4.24 + 4.38 + 4.68)$

$= 0.91 + 0.01(20.29) = 1.1129$

$PM_{nominal} = A * (size)^B$

$= 2.5 * (50)^{1.1129} = 194.41$ Person months.

The 7 cost drivers are

PDIF = high (1.29)

PERS = high (0.83)

RCPX = nominal (1.0)

RUSE = nominal (1.0)

PREX = nominal (1.0)

FCIL = nominal (1.0)

SCEO = nominal (1.0)

$$PM_{adjusted} = PM_{nominal} * \left[\prod_{i=1}^{7} EM_i\right]$$

$= 194.41 * [1.29 \times 0.83] = 194.41 \times 1.07$

$= 208.155$ Person-months.

4.5.3 Post Architecture Model

The Post architecture Model is the most detailed estimation model and is intended to be used when a software life cycle architecture has been completed. This model is used in the development and maintenance of software products in the application generators, system integration or infrastructure sectors.

The Post Architecture model adjusts nominal effort using 17 efforts multipliers. The large number of multipliers takes advantage of the greater knowledge available later in the development stage.

$$PM_{adjusted} = PM_{nominal} \times \left[\prod_{i=1}^{17} EM_i\right]$$

EM : Effort multiplier which is the product of 17 cost drivers.

The 17 cost drivers of the Post Architecture model are described in the Table 4.16.

Table 4.16: Post Architecture Cost Driver rating level summary

Cost driver	*Purpose*	*Very low*	*Low*	*Nominal*	*High*	*Very high*	*Extra high*
RELY (Reliability required)	Measure of the extent to which the software must perform its intended function over a period of time	Only slight inconvenience	Low, easily recoverable losses	Moderate, easily recoverable losses	High financial loss	Risk to human life	—
DATA (Data base size)	Measure the affect of large data requirements on product development	—	$\frac{\text{Database size(D)}}{\text{Prog. size (P)}} < 10$	$10 \le \frac{D}{P} < 100$	$100 \le \frac{D}{P} < 1000$	$\frac{D}{P} \ge 1000$	—
CPLX (Product complexity)	Complexity is divided into five areas: Control operations, computational operations, device dependent operations, data management operations & User Interface management operations.	See Table 4.17					
RUSE	Measure the required reusability	—	None	Across project	Across program	Across product line	Across multiple product line
DOCU Documentation	Suitability of the project's documentation to its life cycle needs	Many life cycle needs uncovered	Some needs uncovered	Adequate	Excessive for life cycle needs	Very Excessive	—

(Contd...)

TIME (Execution Time constraint)	Measure of execution time constraint on software	—	—	≤ 50% use of a available execution time	70%	85%	95%
STOR (Main storage constraint)	Measure of main storage constraint on software	—	—	≤ 50% use of available storage	70%	85%	95%
PVOL (Platform Volatility)	Measure of changes to the OS, compilers, editors, DBMS etc.	—	Major changes every 12 months & minor changes every 1 month	Major: 6 months Minor: 2 weeks	Major: 2 months Minor: 1 week	Major: 2 week Minor: 2 days	—
ACAP (Analyst capability)	Should include analysis and design ability, efficiency & thoroughness, and communication skills.	15th Percentile	35th Percentile	55th Percentile	75th Percentile	90th Percentile	—
PCAP (Programmers capability)	Capability of Programmers as a team. It includes ability, efficiency, thoroughness & communication skills	15th Percentile	35th Percentile	55th Percentile	75th Percentile	90th Percentile	—
PCON (Personnel Continuity)	Rating is in terms of Project's annual personnel turnover	48%/year	24%/year	12%/year	6%/year	3%/year	—
AEXP (Applications Experience)	Rating is dependent on level of applications experience.	≤ 2 months	6 months	1 year	3 year	6 year	—
PEXP (Platform experience)	Measure of Platform experience	≤ 2 months	6 months	1 year	3 year	6 year	—

(Contd...)

LTEX (Language & Tool experience)	Rating is for Language & tool experience	≤ 2 months	6 months	1 year	3 year	6 year	—
TOOL (Use of software tools)	It is the indicator of usage of software tools	No use	Beginning to use	Some use	Good use	Routine & habitual use	—
SITE (Multisite development)	Site location & Communication technology between sites	International with some phone & mail facility	Multicity & multi company with individual phones, FAX	Multicity & multi company with Narrow band mail	Same city or Metro with wideband electronic communication	Same building or complex with wideband electronic communication & Video conferencing	Fully co-located with interactive multimedia
SCED (Required Development Schedule)	Measure of Schedule constraint. Ratings are defined in terms of percentage of schedule stretch-out or acceleration with respect to nominal schedule	75% of nominal	85%	100%	130%	160%	—

Product complexity is based on control operations, computational operations, device dependent operations, data management operations and user interface management operations. Module complexity ratings are given in Table 4.17.

The numeric values of these 17 cost drivers are given in Table 4.18 for the calculation of the product of efforts *i.e.,* effort multiplier (EM). Hence PM adjusted is calculated which will be a better and fine tuned value of effort in person months.

Table 4.17: Module complexity ratings

	Control operations	*Computational operations*	*Device-dependent operations*	*Data management operations*	*User interface management operations*
Very low	Straight-line code with a few non-nested structured programming operators: DOs, CASEs, IF THEN ELSEs. Simple module composition via procedure calls or simple scripts.	Evaluation of simple expressions : *e.g., A = B + C* (D – E)*	Simple read, write statements with simple formats.	Simple arrays in main memory. Simple COTS-DB queries, updates.	Simple input forms, report generators.
Low	Straight forward nesting of structured programming operators. Mostly simple predicates	Evaluation of moderate-level expressions: *e.g., D = SQRT(B**2-4*A*C)*	No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level.	Single file subsetting with no data structure changes, no edits, no intermediate files. Moderately complex COTS-DB queries, updates.	Use of simple graphic user interface (GUI) builders.
Nominal	Mostly simple nesting. Some inter module control Decision tables. Simple callbacks or message passing, including middleware-supported distributed processing.	Use of standard maths and statistical routines. Basic matrix/ vector operations.	I/O processing includes device selection, status checking and error processing.	Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates.	Simple use of widget set.

(Contd.)...

High	Highly nested structured programming operators with many compound predicates. Queue and stack control. Homogeneous, distributed processing. Single processor soft real-time control.	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, roundoff concerns.	Operations at physical I/O level (physical storage address translations; seeks, read, etc.) Optimized I/O overlap.	Simple triggers activated by data stream contents. Complex data restructuring.	Widget set development and extension. Simple voice I/O, multimedia.
Very high	Reentrant and recursive coding. Fixed-priority interrupt handling. Task synchronization, complex callbacks, heterogeneous distributed processing. Single-processor hard real-time control.	Difficult but structured numerical analysis: near-singular matrix equations, partial differential equations. Simple parallelization.	Routines for interrupt diagnosis, servicing, masking. Communication line handling. Performance-intensive embedded systems.	Distributed database coordination. Complex triggers. Search optimization.	Moderately complex 2D/3D, dynamic graphics, multimedia.
Extra high	Multiple resource scheduling with dynamically changing priorities. Microcode-level control. Distributed hard real-time control.	Difficult and unstructured numerical analysis: highly accurate analysis of noisy, stochastic data. Complex parallelization.	Device timing-dependent coding, micro-programmed operations. Performance-critical embedded systems.	Highly coupled, dynamic relational and object structures. Natural language data management.	Complex multimedia, virtual reality.

Table 4.18: 17 Cost drivers

Cost driver	Rating					
	Very low	*Low*	*Nominal*	*High*	*Very high*	*Extra high*
RELY	0.75	0.88	1.00	1.15	1.39	
DATA		0.93	1.00	1.09	1.19	
CPLX	0.75	0.88	1.00	1.15	1.30	1.66
RUSE		0.91	1.00	1.14	1.29	1.49
DOCU	0.89	0.95	1.00	1.06	1.13	
TIME			1.00	1.11	1.31	1.67
STOR			1.00	1.06	1.21	1.57
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.50	1.22	1.00	0.83	0.67	
PCAP	1.37	1.16	1.00	0.87	0.74	
PCON	1.24	1.10	1.00	0.92	0.84	
AEXP	1.22	1.10	1.00	0.89	0.81	
PEXP	1.25	1.12	1.00	0.88	0.81	
LTEX	1.22	1.10	1.00	0.91	0.84	
TOOL	1.24	1.12	1.00	0.86	0.72	
SITE	1.25	1.10	1.00	0.92	0.84	0.78
SCED	1.29	1.10	1.00	1.00	1.00	

Schedule estimation

Development time can be calculated using $PM_{adjusted}$ as a key factor and the desired equation is:

$$TDEV_{nominal} = [\phi \times (PM_{adjusted})^{(0.28 + 0.2(B - 0.091))}] * \frac{SCED\ \%}{100}$$

where ϕ = constant, provisionally set to 3.67

$TDEV_{nominal}$ = calendar time in months with a scheduled constraint

B = Scaling factor

$PM_{adjusted}$ = Estimated effort in Person months (after adjustment)

Size measurement

Size can be measured in any unit and the model can be calibrated accordingly. However, COCOMO II details are:

(*i*) Application composition model uses the size in object points.

(*ii*) The other two models use size in KLOC.

Early design model uses unadjusted function points. These function points are converted into KLOC using Table 4.19. Post architecture model may compute KLOC after defining LOC counting rules. If function points are used, then use unadjusted function points and convert it into KLOC using Table 4.19 [JUNE 91].

Table 4.19: Converting function points to lines of code

Language	*SLOC/UFP*
Ada	71
AI Shell	49
APL	32
Assembly	320
Assembly (Macro)	213
ANSI/Quick/Turbo Basic	64
Basic-Compiled	91
Basic-Interpreted	128
C	128
C++	29
ANSI Cobol 85	91
Fortan 77	105
Forth	64
Jovial	105
Lisp	64
Modula 2	80
Pascal	91
Prolog	64
Report Generator	80
Spreadsheet	6

COCOMO II reflects the experience and data collection of developers. It is a complex model and there are many attributes with too much scope for uncertainty in estimating their values. Each organisation should calibrate the model and attribute values according to its own historical data which may reflect local circumstances that affect the model.

Example 4.11

Consider the software project given in example 4.10. Size and scale factor (B) are the same. The identified 17 Cost drivers are high reliability (RELY), very high database size (DATA), high execution time constraint (TIME), very high analyst capability (ACAP), high programmers capability (PCAP). The other cost drivers are nominal. Calculate the effort in Person-Months for the development of the project.

Solution

Here B = 1.1129

$PM_{nominal}$ = 194.41 Person-months

$$PM_{adjusted} = PM_{nominal} \times \left[\prod_{i=1}^{17} EM_i\right]$$

$$= 194.41 \times (1.15 \times 1.19 \times 1.11 \times 0.67 \times 0.87)$$

$$= 194.41 \times 0.885$$

$$= 172.05 \text{ Person-months.}$$

If analyst capability is very high alongwith high programmers capability, the effort will be reduced significantly. If such human resource factors are low, effort will be increased drastically. Therefore, human resource factors should be considered very carefully and are very important for the success of the project. If we consider estimated adjusted effort ($PM_{adjusted}$) of both the cases, the values are 208.155 Person-months and 172.05 Person-months. Hence difference is of 36.105 Person-months. More we know, better is the estimate, hence, effort estimated in Post Architecture model is more realistic and reasonable.

4.6 THE PUTNAM RESOURCE ALLOCATION MODEL

Norden [NORD58] of IBM observed that the Rayleigh curve can be used as an approximate model for a range of hardware development projects. This approach was later extended by Putnam to apply to software projects. Putnam observed that the Rayleigh curve (Fig. 4.6) was a close representation, not only at the project level but also for software subsystem development. As many as 150 projects were studied by Norden [NORD77] and subsequently by Putnam, and apparently both researchers observed the same tendency for the manpower curve to rise, peak, and then exponentially trail off as a function of time.

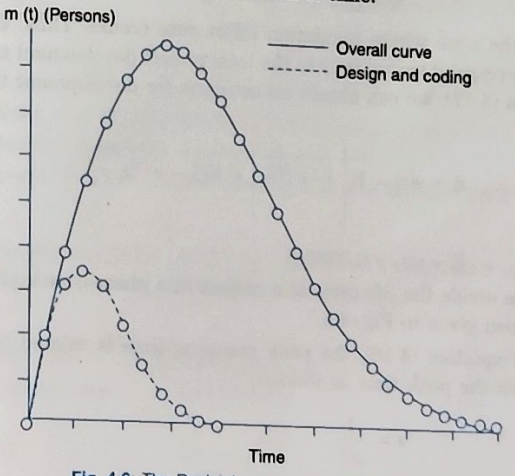

Fig. 4.6: The Rayleigh manpower loading curve

4.6.1 The Norden/Rayleigh Curve

The Norden/Rayleigh equation represents manpower, measured in persons per unit time as a function of time. It is usually expressed in person-year/year (PY/YR). The Rayleigh curve is modeled by the differential equation

$$m(t) = \frac{dy}{dt} = 2Kate^{-at^2} \tag{4.16}$$

where dy/dt is the manpower utilization rate per unit time, "t" is elapsed time, "a" is a parameter that affects the shape of the curve, and "K" is the area under the curve in the interval $[0, \infty]$. Integrating equation (4.16), on interval $[0, t]$, we obtain

$$y(t) = K[1 - e^{-at^2}] \tag{4.17}$$

where $y(t)$ is the cumulative manpower used upto time t.

$$y(0) = 0$$
$$y(\infty) = K$$

The cumulative manpower is null at the start of the project, and grows monotonically towards the total effort K (area under the curve).

It can be seen from the Fig. 4.7 that the parameter "a", which has the dimensions of 1/time^2, plays an important role in the determination of the peak manpower. The larger the value of "a", earlier the peak time occurs and steeper is the person profile. By deriving the manpower function relative to time and finding the zero value of this derivative, the relationship between the peak time, "t_d", and "a" can be found to be:

$$\frac{d^2y}{dt^2} = 2Kae^{-at^2}\left[1 - 2at^2\right] = 0$$

$$\therefore \quad t_d^2 = \frac{1}{2a} \tag{4.18}$$

"t_d" denotes the time where maximum effort rate occurs. Thus, the point "t_d" on the time scale should correspond very closely to the total project development time. If we substitute "t_d" for t in equation (4.17), we can obtain an estimate for development time

$$E = y(t) = K\left(1 - e^{-\frac{t_d^2}{2t_d^2}}\right) = K(1 - e^{-0.5})$$

$$E = y(t) = 0.3935\ K \tag{4.19}$$

Actually if we divide the life cycle of a project into phases, each phase can be modeled by a curve of the form given in Fig. 4.6.

As shown in equation (4.18), the peak manning time is related to "a". Therefore, "a" can be obtained from the peak time as follows:

$$a = \frac{1}{2t_d^2}$$

The number of people involved in the project at the peak time then becomes easy to determine by replacing "a" with $1/2t_d^2$ in the Norden/ Rayleigh model. By making this substitution in equation (4.16), we have

$$m(t) = \frac{2K}{2t_d^2} te^{-\frac{t^2}{2t_d^2}}$$

$$= \frac{K}{t_d^2} te^{-\frac{t^2}{2t_d^2}}$$

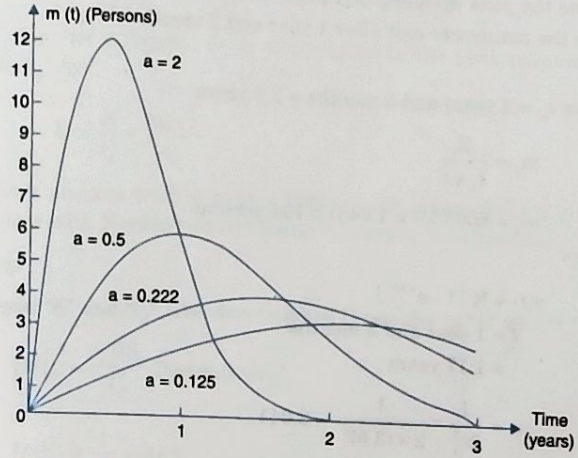

Fig. 4.7 Influence of parameter 'a' on the manpower distribution

At time $t = t_d$, the peak manning, $m(t_d)$ is obtained, which is denoted by m_o. Thus, expression for the peak manning of the project is

$$m_o = \frac{K}{t_d\sqrt{e}}$$

Where "K" is the total project cost (or effort) in person-years, "t_d" is the delivery time in years, "m_o" is the number of persons employed at the peak.

The average rate of software team build-up can also be calculated by dividing m_o by t_d.

Example 4.12

A software project is planned to cost 95 PY in a period of 1 year and 9 months. Calculate the peak manning and average rate of software team build up.

Solution

Software project cost $K = 95$ PY

Peak development time $t_d = 1.75$ years

Peak manning $= m_o = \frac{K}{t_d\sqrt{e}}$

$$\frac{95}{1.75 \times 1.648} = 32.94 \equiv 33 \text{ persons}$$

Average rate of software team build-up

$$= m_o/t_d = 33/1.75 = 18.8 \text{ person/year or } 1.56 \text{ person/month.}$$

Example 4.13

Consider a large-scale project for which the manpower requirement is K = 600 PY and the development time is 3 years 6 months.

(*a*) Calculate the peak manning and peak time.

(*b*) What is the manpower cost after 1 year and 2 months?

Solution

(*a*) We know t_d = 3 years and 6 months = 3.5 years

Now $$m_o = \frac{K}{t_d\sqrt{e}}$$

$$\therefore \quad m_o = 600/(3.5 \times 1.648) \equiv 104 \text{ persons}$$

(*b*) We know

$$y(t) = K\,[1 - e^{-at^2}]$$

t = 1 year and 2 months

= 1.17 years

$$a = \frac{1}{2t_d^2} = \frac{1}{2 \times (3.5)^2} = 0.041$$

$$y(1.17) = 600[1 - e^{-0.041(1.17)^2}]$$

$$= 32.6 \text{ PY}$$

4.6.2 Difficulty Metric

The slope of the manpower distribution at start time (t = 0) also has some useful properties. By differentiating the Norden/Rayleigh function with respect to time, the following equation is obtained.

$$m'(t) = \frac{d^2y}{dt^2} = 2Kae^{-at^2}\,(1 - 2at^2)$$

Then, for $t = 0$,

$$m'(0) = 2Ka = \frac{2K}{2t_d^2} = \frac{K}{t_d^2} \qquad (4.20)$$

The ratio $\frac{K}{t_d^2}$ is called difficulty and is denoted by D, which is measured in person/year:

$$D = \frac{K}{t_d^2} \qquad (4.21)$$

This relationship shows that a project is more difficult to develop when the manpower demand is high or when the time schedule is short (small t_d). It is also interesting to note that difficult projects will tend to have a steeper demand for manpower at the beginning for the same time scale. After studying a large number (about 50) of Army developed software projects, Putnam observed that for systems that were relatively easy to develop, D tended to be small, while for systems that were relatively hard to develop, D tended to be large.

Peak manning is defined as

$$m_0 = \frac{K}{t_d\sqrt{e}}$$

We notice that the difficulty, D, is also related to the peak manning, "m_0" and the development time "t_d" by

$$D = \frac{K}{t_d^2} = \frac{m_0\sqrt{e}}{t_d}$$

Thus, difficult projects tend to have a higher peak manning for a given development time, which is in line with Norden's observations relative to the parameter "a" [LOND87].

Manpower buildup

D is dependent upon "K" and "t_d". The derivative of D relative to "K" and "t_d" are:

$$D'(t_d) = \frac{-2K}{t_d^3} \text{ Person/years}^2$$

$$D'(K) = \frac{1}{t_d^2} \text{ year}^{-2}$$

In practice, $D^1(K)$ will always be very much smaller than the absolute value of $D'(t_d)$. This difference in sensitivity is shown by considering two projects

Project A : Cost = 20 PY & t_d = 1 year

Project B : Cost = 120 PY & t_d = 2.5 years

The derivative values are

Project A : $D'(t_d) = -40$ & $D'(K) = 1$

Project B : $D'(t_d) = -15.36$ & $D'(K) = 0.16$

This shows that a given software development is time sensitive.

Putnam also observed that the difficulty derivative relative to time played an important role in explaining the behaviour of software development. He noted that if the project scale is increased the development time also increases to such an extent that the quantity K/t_d^3 remains constant around a value, which could be 8, 15 or 27. This quantity is represented by D_0 and can be expressed as:

$$D_0 = \frac{K}{t_d^3} \text{ Person/year}^2$$

The value of D_0 is related to the nature of software developed in the following way:

- $D_0 = 8$ refers to entirely new software with many interfaces and interactions with other systems.
- $D_0 = 15$ refers to new stand alone system.
- $D_0 = 27$ refers to the software that is rebuilt from existing software.

Putnam also discovered that D_0 could vary slightly from one organisation to another depending on the average skill of the analysts, developers and the management involved.

In practice, D_0 has a strong influence on the shape of the manpower distribution. The larger D_0 is, the steeper manpower distribution is, and the faster the necessary manpower build up will be. For this reason, the quantity D_0 is called the manpower build up.

Example 4.14

Consider the example 4.13 and calculate the difficulty and manpower build up.

Solution

We know

Difficulty $$D = \frac{K}{t_d^2}$$

$$= \frac{600}{(3.5)^2} = 49 \text{ person/year}$$

Manpower build up can be calculated by following equation

$$D_0 = \frac{K}{t_d^3}$$

$$= \frac{600}{(3.5)^3} = 14 \text{ person/year}^2.$$

4.6.3 Productivity Versus Difficulty

It is appropriate to find relationship between productivity and difficulty. Productivity is defined as the number of lines of code developed per person-month. Putnam has observed that productivity is proportional to the difficulty

$$P \propto D^{\beta} \tag{4.22}$$

The average productivity may be defined as :

P = Lines of code produced / Cumulative manpower used to produce code

$$P = S/E \tag{4.23}$$

where S is lines of code produced and E is cumulative manpower used from $t = 0$ to $t = t_d$ (inception of the project to the delivery time).

Using non-linear regression, Putnam determined from an analysis of 50 army projects that

$$P = \phi D^{-2/3} \tag{4.24}$$

Using equation 4.23, this relationship may be written as

$$S = \phi D^{-2/3} E$$

$$= \phi D^{-2/3} (0.3935 K)$$

Using equation 4.21, we have

$$S = \phi \left[\frac{K}{t_d^2}\right]^{-\frac{2}{3}} K(0.3935)$$

$$S = 0.3935\ \phi K^{1/3} t_d^{4/3} \tag{4.25}$$

In the usual form of this expression, the quantity $0.3935\ \phi$ is replaced by a co-efficient C, which is given the name of Technology Factor. It reflects the effect of various factors on productivity such as hardware constraints, program complexity, personnel experience levels, and the programming environment. Putnam has proposed using a discrete spectrum of 20 values for C ranging from 610 to 57314 (assuming that K is measured in person-years and T in years) depending on an assessment of the technology factor that applies to the project under consideration. Equation 4.25 may be modified and now written as:

$$S = CK^{1/3} t_d^{4/3} \tag{4.26}$$

The value of C can also be found out as:

$$C = S.K^{-1/3} t_d^{-4/3} \tag{4.27}$$

It is easy to use the size, cost and development time of past projects to determine the value of C and hence to revise the value of C obtained to model forth coming projects.

4.6.4 The Trade-off between Time Versus Cost

In software projects, time cannot be freely exchanged against cost. Such a trade off is limited by the nature of software development. For a given organisation, developing a software of size S, the quantity obtained from equation 4.26 is constant. Using equation 4.26, we have

$$K^{1/3} t_d^{4/3} = S/C$$

If we raise power by 3, then Kt_d^4 is constant for a constant size software. A compression of the development time t_d will produce an increase of manpower cost. If compression is excessive, not only would the software development cost much more, but also the development would become so difficult that it would increase the risk of being unmanageable. This is in line with a remark made by Boehm that the time scale should never be reduced to less than 75% of its initial calculated value [LOND87].

The name given by Putnam to the later versions of this model is Software Life cycle Methodology (SLIM). This model is a combination of expertise and statistical computations and could be used effectively for predictive purposes if we had a suitable algorithm that we might use to predict the value of C for a software project. Using equation 4.27, we have

$$K = \frac{1}{t_d^4}\left[\frac{S}{C}\right]^3 \tag{4.28}$$

For a software product of a given size and fixed development environment, equation (4.28) implies that the effort K varies inversely as the fourth power of the development time. For instance, if we take the constant C to be 5000 and if we estimate the size of the project S = 500,000 LOC then

$$K = \frac{1}{t_d^4}(100)^3$$

Table 4.20 shows how the required effort in person-years changes as the development time measured in years changes. Thus, reducing the development time from 5 years to 4 years would increase the total effort and the cost by a factor 2.4, reducing it to 3 years would increase them by a factor of 7.7.

Table 4.20: Manpower versus development time

t_d (years)	K (person-years)
5.0	1600
4.0	3906
3.5	6664
3.0	12346

Putnam attempted to offer support for his use of "t_d^4" in equation (4.28) after examining 750 software systems [PUTN84]. With 251 of them it was shown that equation (4.28) is an acceptable model of the relationship among "K", "S", and "t_d". However C was computed using equation 4.27; it was not the result of some independent assessment of the technology level. Therefore, the data from 251 systems given in [PUTN84] may be said to offer only marginal support for the fourth power law. Furthermore, there was no evidence offered that the same relationship holds for the other 499 systems.

4.6.5 Development Sub-cycle

All that has been described so far is related to the project life cycle, as represented in Fig. 4.8, by the project curve.

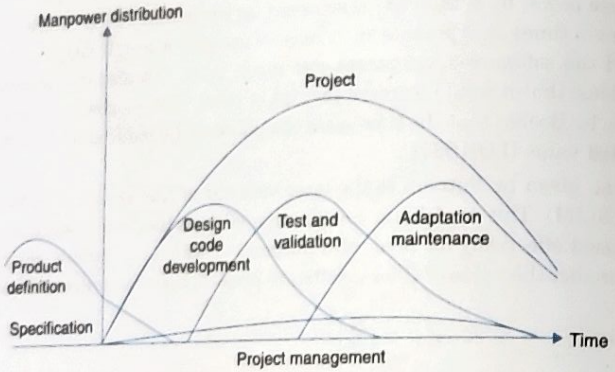

Fig. 4.8: Project life cycle

Curve is represented by a Rayleigh function, which gives the manning level relative to time and reaches a peak at time t_d. The project curve is the addition of two curves called development curve and test and validation curve. Both the curves are the sub-cycles of the project curve and can be modeled by Rayleigh function.

Let $m_d(t)$ and $y_d(t)$ be the design manning and the cumulative design manpower cost which can be represented by:

$$m_d(t) = 2K_d bt\, e^{-bt^2} \tag{4.29}$$

$$y_d(t) = K_d[1 - e^{-bt^2}] \tag{4.30}$$

An examination of $m_d(t)$ function shows a non-zero value for m_d at time "t_d". This is because the manpower involved in design and coding is still completing this activity after "t_d" in the form of rework due to the validation of the product. Nevertheless, for the model a level of completion has to be assumed for development.

It is good practical assumption to assume that the development, will be 95% completed by the time t_d, this gives

$$\frac{y_d(t)}{K_d} = 1 - e^{-bt^2} = 0.95 \tag{4.31}$$

It is then legitimate by set of previous definitions and by analogy with the Norden co-efficient "a", to set:

$$b = \frac{1}{2t_{od}^2} \tag{4.32}$$

Where t_{od} is the time at which the development curve exhibits a peak manning. This can then be used to obtain the following relation between the development time "t_d", and development peak manning, t_{od}:

$$t_{od} = \frac{t_d}{\sqrt{6}} \tag{4.33}$$

Relationship between "K_d" (total manpower cost of the development sub-cycle) and K (total manpower cost of the generic cycle) must be established. This can be obtained by using the observation that at the origin of time both cycles have the same slope. Thus from equation (4.21) and by differentiation of equation (4.29)

$$\left(\frac{dm}{dt}\right)_o = \frac{K}{t_d^2} = \frac{K_d}{t_{od}^2} = \left(\frac{dm_d}{dt}\right)_o$$

Consider equation (4.33), and we have

$$K_d = K/6 \tag{4.34}$$

It should also be noted that the difficulty D, is the same whether expressed in terms of "K" and "t_d" or "K_d" and "t_{od}". More formally, this can be stated by:

$$D = \frac{K}{t_d^2} = \frac{K_d}{t_{od}^2}$$

This does not apply to the manpower build up D_o

$$D_o = \frac{K}{t_d^3} = \frac{K_d}{\sqrt{6}\, t_{od}^3} \tag{4.35}$$

Note here that there is an extra factor $\sqrt{6}$ when the calculations are made at the sub cycle level.

Conte et al., [CONT86] investigated the Putnam models and observed that they work reasonably well on very large systems, but seriously over estimate effort on medium or small size systems. The model emphasises heavily on the size and development schedule attributes while down playing with other attributes. The constant C must be used to reflect all other attributes including complexity, use of modern programming practices and personnel ability.

Example 4.15

A software development requires 90 PY during the total development sub-cycle. The development time is planned for a duration of 3 years and 5 months [CONT86]

(*a*) Calculate the manpower cost expended until development time.

(*b*) Determine the development peak time

(*c*) Calculate the difficulty and manpower build-up.

Solution

(*a*) Duration t_d = 3.41 years

We know from equation (4.31)

$$\frac{y_d(t_d)}{K_d} = 0.95$$

$$Y_d(t_d) = 0.95 \times 90$$

$$= 85.5 \text{ PY}$$

(*b*) We know from equation (4.33)

$$t_{od} = \frac{t_d}{\sqrt{6}} = 3.41/2.449 = 1.39 \text{ years}$$

$$\equiv 17 \text{ months.}$$

(*c*) Total Manpower development

$$K_d = y_d(t_d)/0.95$$

$$= 85.5/0.95 = 90$$

$$K = 6K_d = 90 \times 6 = 540 \text{ PY}$$

$$D = K/t_d^2 = 540/(3.41)^2 = 46 \text{ person/years}$$

$$D_0 = \frac{K}{t_d^3} = 540/(3.41)^3 = 13.6 \text{ person/year}^2.$$

Example 4.16

A software development for avionics has consumed 32 PY upto development cycle and produced a size of 48000 LOC. The development of project was completed in 25 months. Calculate the development time, total manpower requirement, development peak time, difficulty, manpower build up and technology factor.

Solution

Development time t_d = 25 months = 2.08 years

Total manpower development $K_d = \frac{Y_d(t_d)}{0.95}$

$$K_d = \frac{32}{0.95} = 33.7 \text{ PY}$$

Development peak time $t_{od} = \frac{(t_d)}{\sqrt{6}}$

$$= 0.85 \text{ years (or 10 months).}$$

$$K = 6K_d = 6 \times 33.7 = 202 \text{ PY}$$

$$D = \frac{K}{t_d^2} = \frac{202}{(2.08)^2} = 46.7 \text{ person/year}$$

$$D_0 = \frac{K}{t_d^3} = \frac{202}{(2.08)^3} = 22.5 \text{ person/year}^2$$

Technology factor

$$C = SK^{-1/3}t_d^{-4/3}$$

$$= 48000 \times (202)^{-1/3} (2.08)^{-4/3}$$

$$= 3077.$$

Example 4.17

What amount of software can be delivered in 1 year 10 months in an organisation whose technology factor is 2400 if a total of 25 PY is permitted for development effort?

Solution

$$t_d = 1.8 \text{ years}$$

$$K_d = 25 \text{ PY}$$

$$K = 25 \times 6 = 150 \text{ PY}$$

$$C = 2400$$

We know $S = CK^{1/3} t_d^{4/3}$

$$= 2400 \times 5.313 \times 2.18 = 27920 \text{ LOC}$$

Example 4.18

The software development environment of an organisation developing real time software has been assessed at technology factor of 2200. The maximum value of manpower build up for this type of software is D_o = 7.5. The estimated size of the software to be developed is S = 55000 LOC [LOND87].

(*a*) Determine the total development time, the total development manpower cost, the difficulty and the development peak manning.

(*b*) The development time determined in (*a*) is considered too long. It is recommended that it be reduced by two months. What would happen?

Solution

We have $S = CK^{1/3}t_d^{4/3}$

$$\left(\frac{S}{C}\right)^3 = Kt_d^4$$

which is also equivalent to

$$\left(\frac{S}{C}\right)^3 = D_o t_d^7$$

then $$t_d = \left[\frac{1}{D_0}\left(\frac{S}{C}\right)^3\right]^{1/7}$$

Since $\frac{S}{C} = 25$,

$t_d = 3$ years

As $K = D_0 t_d^3 = 7.5 \times 27 = 202$ PY

Total development manpower cost $K_d = \frac{202}{06} = 33.75$ PY

$$D = D_0 t_d = 22.5 \text{ person/year}$$

$$t_{od} = \frac{t_d}{\sqrt{6}} = \frac{3}{\sqrt{6}} = 1.2 \text{ years}$$

Using equations (4.20) and (4.29), we have

$$m_d(t) = 2K_d\, bte^{-bt^2}$$

$$Y_d(t) = K_d(1 - e^{-bt^2})$$

Here, $t = t_{od}$

Peak manning $= m_{od} = Dt_{od}e^{-1/2}$

$= 22.5 \times 1.2 \times .606 \cong 16$ persons

(*b*) Developing time reduction means either developing the software at a higher manpower build-up or producing less software.

(*i*) **Increase manpower build-up**

$$D_o = \frac{1}{t_d^7}\left(\frac{S}{C}\right)^3$$

The new development time would be 2.8 years and new manpower build-up is

$D_o = (25)^3/(2.8)^7 = 11.6$ person/year^2

$K = D_0 t^3{}_d = 254$ PY

$$K_d = \frac{254}{6} = 42.4 \text{ PY}$$

$$D = D_0 t_d = 32.5 \text{ person/year}$$

The peak time is $t_{od} = 1.14$ years

Peak manning $m_{od} = Dt_{od}\, e^{-0.5}$

$= 32.5 \times 1.14 \times 0.6 \cong 22$ persons

Note the huge increase in peak manning and manpower cost.

(*ii*) **Produce less software**

$$\left(\frac{S}{C}\right)^3 = D_0\, t_d^7 = 7.5 \times (2.8)^7 = 10119.696$$

$$\left(\frac{S}{C}\right)^3 = 21.62989$$

Then for $C = 2200$

$S = 47586$ LOC

The problem is now to decide which software functions can be cut down.

Example 4.19

A stand-alone project for which the size is estimated at 12500 LOC is to be developed in an environment such that the technology factor is 1200. Choosing a manpower build up $D_o = 15$, calculate the minimum development time, total development man power cost, the difficulty, the peak manning, the development peak time, and the development productivity.

Solution

Size (S) = 12500 LOC

Technology factor (C) = 1200

Manpower build up (D_o) = 15

Now $S = CK^{1/3}t_d^{4/3}$

$$\frac{S}{C} = K^{1/3}t_d^{4/3}$$

$$\left(\frac{S}{C}\right)^3 = Kt_d^4$$

Also we know $D_o = \frac{K}{t_d^3}$

$K = D_o t_d^3 = D_o t_d^3$

Hence $\left(\frac{S}{C}\right)^3 = D_o t_d^7$

Substituting the values, we get

$$\left(\frac{12500}{1200}\right)^3 = 15t_d^7$$

$$t_d = \left[\frac{(10.416)^3}{15}\right]^{1/7}$$

$t_d = 1.85$ years

(*i*) Hence Minimum development time (t_d) = 1.85 years.

(*ii*) Total development manpower cost $K_d = \frac{K}{6}$

Hence, $K = 15t_d^3$

$= 15(1.85)^3 = 94.97$ PY

$K_d = \frac{K}{6} = \frac{94.97}{6} = 15.83$ PY

(*iii*) Difficulty $D = \frac{K}{t_d^2} = \frac{94.97}{(1.85)^2} = 27.75$ Person/year

(*iv*) Peak Manning $m_0 = \frac{K}{t_d\sqrt{e}}$

$= \frac{94.97}{1.85 \times 1.648} = 31.15$ Persons

(*v*) Development Peak time $t_{od} = \frac{t_d}{\sqrt{6}}$

$= \frac{1.85}{2.449} = 0.755$ years

(*vi*) Development Productivity

$= \frac{\text{No. of lines of code (S)}}{\text{effort } (K_d)}$

$= \frac{12500}{15.83} = 789.6$ LOC/PY.

4.7 SOFTWARE RISK MANAGEMENT

We, software developers are extremely optimists. When planning software projects, we often assume that everything will go exactly as planned. Alternatively, we take the other extreme position. The creative nature of software development means we can never accurately predict what is going to happen, so what is the point of making detailed plans? Both these perspectives can lead to software surprises, when unexpected things happen that throw the project completely off track. Software surprises are never good news.

Risk Management is becoming recognised as an important area in the software industry to reduce this surprise factor. Risk management means dealing with a concern before it becomes a crisis. Therefore, most of the software development activities include risk management as a key part of the planning process and expect the plan to highlight the specific risk areas. The project planning is expected to quantify both probability of failure and consequences of failure and to describe what will be done to reduce the risk.

4.7.1 What is Risk?

Tomorrow's problems are today's risks. Hence, a simple definition of a "risk" is a problem that could cause some loss or threaten the success of the project, but which has not happened yet.

These potential problems might have an adverse impact on cost, schedule, or technical success of the project, the quality of our software products, or project team morale. Risk management is the process of identifying, addressing and eliminating these problems before they can damage the project. ① ② ③

We need to differentiate risks, as potential problems, from the current problems of the project. Different approaches are required to address these two kinds of issues. For example, a staff shortage because we have not been able to hire people with the right technical skills is a current problem; but the threat of our technical people being hired away by the competition is a risk. Current real problems require prompt, corrective action, whereas risk can be dealt with in several different ways. We might choose to avoid the risk entirely by changing the project approach or even cancelling the project.

Whether we tackle them head-on or keep our heads in the sand, risks have a potentially huge impact on many aspects of the project. We do far too much pretending in software. We pretend, we know who our users are, we know what their needs are, that we would not have staff turn over problems, that we can solve all technical problems that arise, that our estimates are achievable, and that nothing unexpected will happen.

Risk management is about discarding the rose-coloured glasses and confronting the very real potential of undesirable events conspiring to throw our project off track [WIEG98].

4.7.2 Typical Software Risks

The list of evil things that can befall a software project is depressingly long. Possible risks can come from group brainstorming activities, or from a risk factor chart accumulated from previous projects. There are no magic solutions to any of these risk factors, so we need to rely on past experience and a strong knowledge of contemporary software engineering and management practices to control these risks. Capers Jones has identified the top five risk factors that threaten projects in different applications [JONE94].

Dependencies

Many risks arise due to dependencies of project on outside agencies or factors. It is not easy to control these external dependencies. Some typical dependency-related risk factors are:

- Availability of trained, experienced people
- Intercomponent or inter-group dependencies
- Customer-furnished items or information
- Internal and external subcontractor relationships

Requirement Issues

Many projects face uncertainty and turmoil around the product's requirements. While some of this uncertainty is tolerable in early stages, but the threat to success increases if such issues are not resolved as the project progresses. If we do not control requirements-related risk factors, we might either build the wrong product, or build the right product badly. Either situation results in unpleasant surprises and unhappy customers. Some typical factors are:

- Lack of clear product vision
- Lack of agreement on product requirements

strong knowledge

- Unprioritized requirements
- New market with uncertain needs
- Rapidly changing requirements
- Inadequate impact analysis of requirements changes

Management issues

Project Managers usually write the risk management plan, and most people do not wish to air their weaknesses (assuming they even recognise them) in public. Nonetheless, issues like those listed below can make it harder for projects to succeed. If we do not confront such touchy issues, we should not be surprised if they bite us at some point. Defined project tracking processes, and clear roles and responsibilities, can address some of these risk factors.

- Inadequate planning and task identification
- Inadequate visibility into actual project status
- Unclear project ownership and decision making
- Unrealistic commitments made, sometimes for the wrong reasons
- Managers or customers with unrealistic expectations
- Staff personality conflicts
- Poor communication

Lack of knowledge

The rapid rate of change of technologies, and the increasing change of skilled staff, mean that our project teams may not have the skills we need to be successful. The key is to recognise the risk areas early enough so that we can take appropriate preventive actions, such as obtaining training, hiring consultants, and bringing the right people together on the project team. Some of the factors are:

- Inadequate training
- Poor understanding of methods, tools, and techniques
- Inadequate application domain experience
- New technologies
- Ineffective, poorly documented, or neglected processes

Other risk categories

The list of potential risk areas is long. Some of the critical areas are:

- Unavailability of adequate testing facilities
- Turnover of essential personnel
- Unachievable performance requirements
- Technical approaches that may not work

4.7.3 Risk Management Activities

Risk management involves several important steps, each of which is illustrated in Fig. 4.9. We should assess the risks on the project, so that we understand what may occur during the

course of development or maintenance. The assessment consists of three activities: identifying the risks, analysing them, and assigning priorities to each of them.

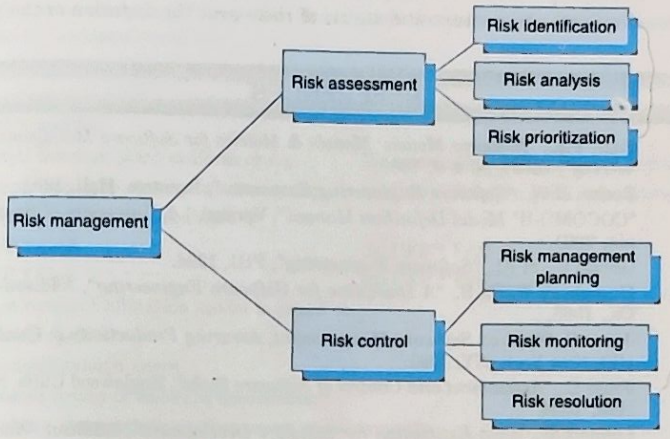

Fig. 4.9: Risk management activities

Risk assessment

It is the process of examining a project and identifying areas of potential risk. Risk identification can be facilitated with the help of a checklist of common risk areas of software projects; or by examining the contents of an organizational database of previously identified risks. Risk analysis involves examining how project outcomes might change with modification of risk input variables. Risk prioritization helps the project focus on its most severe risks by assessing the risk exposure. Exposure is the product of the probability of incurring a loss due to the risk and the potential magnitude of that loss. This prioritization can be done in a quantitative way, by estimating the probability (0.1 – 1.0) and relative loss, on a scale of 1 to 10. Multiplying these factors together provide an estimation of risk exposure due to each risk item, which can run from 0.1 (do not give it another thought) through 10 (stand back, here it comes!). The higher the exposure, the more aggressively the risk should be tackled. It may be easier to simply estimate both probability and impact as High, Medium, or Low. Those items having at least one dimension rated as High are the ones to worry about first.

Another way of handling risk is the risk avoidance. Do not do the risky things! We may avoid risks by not undertaking certain projects, or by relying on proven rather than cutting edge technologies.

Risk control

It is the process of managing risks to achieve the desired outcomes. Risk management planning produces a plan for dealing with each significant risk. It is useful to record decisions in the plan, so that both customer and developer can review how problems are to be avoided, as well as how they are to be handled when they arise. We should also monitor the project as

development progresses, periodically revaluating the risks, their probability, and likely impact. Risk resolution is the execution of the plans for dealing with each risk.

Simply identifying the risks of any project is not enough. We should write them down in a way that communicates the nature and status of risks over the duration of the project.

REFERENCES

[BASL80] Basil V.R., "*Resource Models: Models & Metrics for Software Management and Engineering*", IEEE, pp-4–9, 1980.

[BOEH81] Boehm B.W., "*Software Engineering Economics*", Prentice -Hall, 1981.

[UCSD01] "COCOMO-II" *Model Definition Manual*", Version 1.4, University of Southern California, 2001.

[GHEZ94] Ghezzi C., et Al., "*Software Engineering*", PHI, 1994.

[HUMP95] Humphrey Watts S., "*A Discipline for Software Engineering*", Addison-Wesley, Pub. Co., 1995.

[JONE91] Jones C, "*Applied Software Measurement, Assuring Productivity & Quality*", McGraw Hill, New York, NY, 1991.

[JONE94] Jones C., "*Assessment and Control of Software Risks*", Englewood Cliffs, N.J., Prentice-Hall, 1994.

[LOND87] Londeix B., "*Cost Estimation for Software Development*", Addison -Wesley Pub. Co., 1987.

[NORD58] Norden P.V., "*Curve Fitting for a Model of Applied Research and Development Scheduling*", IBM Journal, Research & Development, Vol 3, No.2, PP.232–248, July, 1958.

[NORD77] Norden P.V., "*Project Life Cycle Modeling: Background and Application of the Life Cycle Curves*", US Army Computer Systems Command, 1977.

[PRESS2K] Pressman Roger, "*Software Engineering*", McGraw Hill Pub., 2000.

[PUTN84] Putnam I.H., Putnam D.T., "*A Verification of the Software Fourth Power Trade off Law*", Proc. Of the Int. Soc. of para-metric Analysis 3, 1, May 184, 443–471

[SAGE90] Sage A.P., & Palmer J.D., "*Software System Engineering*", John Wiley & Sons Pub. Co., 1990.

[SOMM96] Summerville Ian, "*Software Engineering*", Addison - Wesley Pub Co., 1996.

[WALS77] Walson C.E. & Felix C.P., "*A Method for Programming Measurement and Estimation*", IBM System Journal, 16(1), pp - 54–73, 1977.

[WIEG98] Wiegers K.E., "*Know Your Enemy: Software Risk Management*", Software Development Magazine, October, 1998.

MULTIPLE CHOICE QUESTIONS

Note: *Choose most appropriate answer of the following questions*

4.1. After the finalisation of SRS, we may like to estimate
(*a*) size (*b*) cost
(*c*) development time (*d*) all of the above.

4.2. Which one is not a size measure for software
(*a*) LOC (*b*) function Count
(*c*) cyclomatic Complexity (*d*) halstead's program length.

4.3. Function count method was developed by
(*a*) B. Beizer (*b*) B. Boehm
(*c*) M. Halstead (*d*) Alan Albrecht.

4.4. Function point analysis (FPA) method decomposes the system into functional units. The total number of functional units are
(*a*) 2 (*b*) 5
(*c*) 4 (*d*) 1.

4.5. IFPUG Stand for
(*a*) initial function point uniform group (*b*) international function point uniform group
(*c*) international function point user group (*d*) initial function point user group.

4.6. Function point can be calculated by
(*a*) UFP * CAF (*b*) UFP * FAC
(*c*) UFP * Cost (*d*) UFP * Productivity.

4.7. Putnam resource allocation model is based on
(*a*) function points
(*b*) Norden/Rayleigh curve
(*c*) Putnam theory of software management
(*d*) Boehm's observations on manpower utilisation rate.

4.8. Manpower buildup for Putnam resource allocation model is
(*a*) K/t_d^2 persons/year^2 (*b*) K/t_d^3 persons/year^2
(*c*) K/t_d^2 persons/year (*d*) K/t_d^3 persons/year.

4.9. COCOMO was developed initially by
(*a*) B.W. Bohem (*b*) Gregg Rothermal
(*c*) B. Beizer (*d*) Rajeev Gupta.

4.10. A COCOMO model is
(*a*) common cost estimation model (*b*) constructive cost estimation model
(*c*) complete cost estimation model (*d*) comprehensive cost estimation model.

4.11. Estimation of software development effort for organic software in COCOMO is
(*a*) $E = 2.4(KLOC)^{1.05}$ PM (*b*) $E = 3.4(KLOC)^{1.06}$ PM
(*c*) $E = 2.0(KLOC)^{1.05}$ PM (*d*) $E = 2.4(KLOC)^{1.07}$ PM.

4.12. Estimation of size for a project is dependent on
(*a*) cost (*b*) schedule
(*c*) time (*d*) none of the above.

4.13. In function point analysis, number of complexity adjustment factors are
(*a*) 10 (*b*) 20
(*c*) 14 (*d*) 12.

4.14. COCOMO-II estimation model is based on
(*a*) complex approach (*b*) algorithmic approach
(*c*) bottom up approach (*d*) top down approach.

4.15. Cost estimation for a project may include
(*a*) software cost (*b*) hardware cost
(*c*) personnel costs (*d*) all of the above.

4.16. In COCOMO model, if project size is typically 2 – 50 KLOC, then which mode is to be selected?
(a) organic (b) semidetached
(c) embedded (d) none of the above.

4.17. COCOMO-II was developed at
(a) university of Maryland (b) university of Southern California
(c) IBM (d) AT & T Bell labs

4.18. Which one is not a Category of COCOMO-II?
(a) end user programming (b) infrastructure sector
(c) requirement sector (d) system Integration.

4.19. Which one is not an infrastructure software?
(a) operating system (b) database management system
(c) compilers (d) result management system.

4.20. How many stages are in COCOMO-II?
(a) 2 (b) 3
(c) 4 (d) 5.

4.21. Which one is not a stage of COCOMO-II?
(a) application composition estimation model
(b) early design estimation model
(c) post architecture estimation model
(d) comprehensive cost estimation model.

4.22. In Putnam resource allocation model, Rayleigh curve is modeled by the equation
(a) $m(t) = 2at\ e^{-at^2}$ (b) $m(t) = 2Kt\ e^{-at^2}$
(c) $m(t) = 2Kat\ e^{-at^2}$ (d) $m(t) = 2Kbt\ e^{-at^2}$.

4.23. In Putnam resource allocation model, technology factor 'C' is defined as
(a) $C = SK^{-1/3}t_d^{-4/3}$ (b) $C = SK^{1/3}t_d^{4/3}$
(c) $C = SK^{1/3}t_d^{-4/3}$ (d) $C = SK^{-1/3}t_d^{4/3}$.

4.24. Risk management activities are divided in
(a) 3 categories (b) 2 categories
(c) 5 categories (d) 10 categories.

4.25. Which one is not a risk management activity?
(a) risk assessment (b) risk control
(c) risk generation (d) none of the above.

EXERCISE

4.1. What are various activities during software project planning?

4.2. Describe any two software size estimation techniques.

4.3. A proposal is made to count the size of 'C' programs by number of semicolons, except those occurring with literal strings. Discuss the strengths and weaknesses to this size measure when compared with the lines of code count.

4.4. Design a LOC counter for counting LOC automatically. Is it language dependent? What are the limitations of such a counter?

4.5. Compute the function point value for a project with the following information domain characteristics.
Number of user inputs = 30
Number of user outputs = 42
Number of user enquiries = 08
Number of files = 07
Number of external interfaces = 6
Assume that all complexity adjustment values are moderate.

4.6. Explain the concept of function points. Why FPs are becoming acceptable in industry?

4.7. What are size metrics? How is function point metric advantageous over LOC metric? Explain.

4.8. Is it possible to estimate software size before coding? Justify your answer with suitable examples.

4.9. Describe the Albrecht's function count method with a suitable example.

4.10. Compute the function point FP for a payroll program that reads a file of employees and a file of information for the current month and prints cheques for all the employees. The program is capable of handling an interactive command to print an individually requested cheque immediately.

4.11. Assume that the previous payroll program is expected to read a file containing information about all the cheques that have been printed. The file is supposed to be printed and also used by the program next time it is run, to produce a report that compares payroll expenses of the current month with those of the previous month. Compute function points for this program. Justify the difference between the function points of this program and previous one by considering how the complexity of the program is affected by adding the requirement of interfacing with another application (in this case, itself).

4.12. Explain the Walson & Felix model and compare with the SEL model.

4.13. The size of a software product to be developed has been estimated to be 22000 LOC. Predict the manpower cost (effort) by Walston-Felix Model and SEL Model.

4.14. A database system is to be developed. The effort has been estimated to be 100 Persons-Months. Calculate the number of lines of code and productivity in LOC/Person-Month.

4.15. Discuss various types of COCOMO mode. Explain the phase wise distribution of effort.

4.16. Explain all the levels of COCOMO model. Assume that the size of an organic software product has been estimated to be 32,000 lines of code. Determine the effort required to develop the software product and the nominal development time.

4.17. Using the basic COCOMO model, under all three operating modes, determine the performance relation for the ratio of delivered source code lines per person-month of effort. Determine the reasonableness of this relation for several types of software projects.

4.18. The effort distribution for a 240 KLOC organic mode software development project is: product design 12%, detailed design 24%, code and unit test 36%, integrate and test 28%. How would the following changes, from low to high, affect the phase distribution of effort and the total effort: analyst capability, use of modern programming languages, required reliability, requirements volatility?

4.19. Specify, design, and develop a program that implements COCOMO. Using reference [BOEH81] as a guide, extend the program so that it can be used as a planning tool.

4.20. Suppose a system for office automation is to be designed. It is clear from requirements that there will be five modules of size 0.5 KLOC, 1.5 KLOC, 2.0 KLOC, 1.0 KLOC and 2.0 KLOC respectively. Complexity, and reliability requirements are high. Programmer's capability and experience is

low. All other factors are of nominal rating. Use COCOMO model to determine overall cost and schedule estimates. Also calculate the cost and schedule estimates for different phases.

4.21. Suppose that a project was estimated to be 600 KLOC. Calculate the effort and development time for each of the three modes *i.e.*, organic, semidetached and embedded.

4.22. Explain the COCOMO-II in detail. What types of categories of projects are identified?

4.23. Discuss the Infrastructure Sector of COCOMO-II.

4.24. Describe various stages of COCOMO-II. Which stage is more popular and why?

4.25. A software project of application generator category with estimated size of 100 KLOC has to be developed. The scale factor (B) has high precedentness, high development flexibility. Other factors are nominal. The cost drivers are high reliability, medium database size, high Personnel capability, high analyst capability. The other cost drivers are nominal. Calculate the effort in Person-months for the development of the project.

4.26. Explain the Putnam resource allocation model. What are the limitations of this model?

4.27. Describe the trade-off between time versus cost in Putnam resource allocation model.

4.28. Discuss the Putnam resource allocation model. Derive the time and effort equations.

4.29. Assuming the Putnam model, with S = 100,000, C = 5000, D_o = 15, Compute development time t_d and manpower development K_d.

4.30. Obtain software productivity data for two or three software development programs. Use several cost estimating models discussed in this chapter. How do the results compare with actual project results?

4.31. It seems odd that cost and size estimates are developed during software project planning—before detailed software requirements analysis or design has been conducted. Why do we think this is done? Are there circumstances when it should not be done?

4.32. Discuss typical software risks. How staff turnover problem affects software projects?

4.33. What are risk management activities? Is it possible to prioritize the risk?

4.34. What is risk exposure? What techniques can be used to control each risk?

4.35. What is risk? Is it economical to do risk management? What is the effect of this activity on the overall cost of the project?

4.36. There are significant risks even in student projects. Analyse a student project and list the risks.