CHAPTER 5

INTEGRATED LIFE CYCLE METHODOLOGY FOR RESOURCE ESTIMATION

"The typical 200-300 percent cost overrun and similar schedule slippage is no longer tolerable when the cost is measured in millions of dollars and most optimistic schedule expectation is three years or more until an operating product is available to run on the machine. Manpower cost and schedule have to be predicted within reasonable engineering limits before investments decisions are made."

Putnam and Wolverton
Qualitative Management (PuW077)
5.1 Introduction

One of the significant aspects of any project is accurate estimation of resources such as time, manpower and money for its scheduling and execution. IP Stadium was completed as scheduled and within budget (Chat83). In case of engineering disciplines the most common practice adopted for resource estimation is as shown in Fig. 5.1.

For software projects also same practice has been recommended (Wolv74) but there are inherent difficulties due to the peculiar nature of software. Civil engineers have been constructing buildings for so many centuries but the computer age began only thirty years ago. Extensive use of computers dates back only twenty five years and use of high level languages only twenty years. The first sizable programming projects were designed in late 1950's. In 1960's some progress was made in developing resource techniques and now we are at a threshold of better understanding of the nature of software projects and their resource estimation. Normally cost estimates are based on the following approaches:

1. Expert judgement
2. Cost-to-win
3. Psyche bidding
4. Algorithmic Analysis
Lot of research is going on currently in this field of software engineering in the development of algorithms which could be applied to all types of software projects (BaZe78). This endeavour of the author is in the same direction. Our thesis is based on Rayleigh curve (Chapter 4) which is applied to the entire life cycle of a software project for resource estimation.

5.2 Software Estimation

Software Project Management depends on the realistic estimation to provide data from which a project may be costed and progress monitored. The history of software development has not been a happy one (PuWo77) because of lack of such data in open literature.

The most common technique has been to estimate the size of the system and then calculate the programmer man-months required to develop the system (Somm82). This simplistic approach arose from conventional engineering disciplines. The concept of 'man-month' was based on the philosophy that work to be done was a simple product of constant manpower times schedule time and that these factors could be controlled and manipulated. Brooks (Broo75) showed that manpower and time are not interchangeable, programmer productivity varies greatly; and system size and complexity have a great influence on the time required to execute a project.
5.3 **Resource Models**

A substantial amount of work has been done by researchers for developing resource models. These models vary in their outputs and the factors used to calculate the estimates (Fig. 5.2).

There are two major approaches for software estimation - micro and macro. Micro-estimation is based on minute details or small pieces of information scaled upwards while macro-estimation is based upon a view of the overall big picture of the software development process and attempts to forecast manpower and schedule time at the start of the design.

### 5.3.1 **Micro-estimation**

This traditional approach starts with fixing the size, starting and duration of each activity. Then adjustments are made for competence of personnel, complexity of the system, uncertainties and other factors. Doty (Doty77) identified more than 100 factors out of which 42 were judged to be significant and 29 were quantified. Once the significant activities are identified, they may be arranged in a CPM/PERT network. Then the length of the longest path is regarded as scheduled time and the no. of paths in parallel is the no. of people or team to be employed. Wolverton has classified these methods into nine categories (Wolv74).

The micro-estimation methods employ empirical approach to identify the activities that are part of development process of a typical project for a software house. With accounting data from past projects, percentage of the effort expended on each activity is determined. This percentage serves as a baseline and
are initially adjusted to meet the expected demand of a new project.

In Wolverton model total effort is divided in five subareas - analysis (20%), design (18.7%), coding (21%), testing (28.3%) and documentation (11.3%). Each of these subarea effort is subdivided again depending upon the activities in subareas. In this way each activity can be staffed according to individual budget allocated. Allocation of time is determined by history and management intution.

Pragmatically, micro-estimation becomes difficult because of vast volumes of details involved in large systems. This difficulty is not serious as long as small programs developed by individuals are concerned. But when software products consist of hundreds of programs, hundreds of thousands of lines of codes and documentation, developed by team of people and several layers of management, estimating difficulties increase exponentially and these methods fail and give unsatisfactory results. Doty (Doty77) reported that most of the errors result from underestimation of size, sometimes by a factor of three.

5.3.2 Macro-estimation

From practical point of view, realizing the inadequacies and complications of micro-estimation, macro-estimation approach is employed. Without going into minute details, this approach gives an overall picture and is pragmatic in nature.
Conceptually, given certain facts about a software project that are available at the beginning this approach generates an expected curve of life cycle effort against time. A series of milestones are located along the curve. Parallel tolerance curves indicate the confidence level of forecast. This approach provides answers to management questions in terms of management parameters.

5.4 Integrated Life Cycle Methodology

5.4.1 Background

The micro-estimation methods assume that the software development process is linear and static. Brooks (Broo75) showed that it is not so. Norden (Nord70) postulated that this process is curvilinear in nature. Putnam (Putn77) further put forward that this process also possesses a random character. The probability theory and statistical laws may be employed, implying that rates, effort, time, and productivity cannot be measured with great precision because these quantities fluctuate about some average which changes with time. So the averages with standard deviations must be used in estimates.

Putnam applied macroscopic approach to development (design, coding, and testing) phases of software life cycle. Phases - requirements analysis, specification and documentation which are vital in the life cycle of a software product are not included. In our investigation, we have considered the entire life cycle (all the seven phases) and we call it 'Integrated Life Cycle Methodology' (ILCM).
This approach lays out the entire effort distribution across time for a software project. This is a dynamic and multivariable model. Dynamic, because it produces a curve that describes the variation of effort and manpower across time and multivariable because it involves more than one parameter.

This approach is mainly for large software projects. As is to be expected from a statistical approach the correlation reduces as the project size reduces and some interpretation is required when applying it quantitatively. However, this can certainly be used qualitatively to highlight certain important relationships. ILCM deals with aggregate behaviour of software projects. It is not meant for estimating the parameters of individual small programs or the work of individual programmers.

As a general rule, ILC may be employed whenever two or more of the following criteria are satisfied:

1. Product size $\geq 5000$ instructions
2. Project duration $\geq 6$ months
3. Personnel working concurrently $\geq 3$.

### 5.4.2 Fundamentals

In our daily life we find many linear/nearly linear processes in which following rule is applicable:

$$\text{Quantity}(Q) = \text{Rate}(R) \times \text{Time}(T) \quad \text{(where R is constant)}$$

In case of people or labour-intensive process this means:

- Effort($\Phi$) = Manpower($M$) * Time($T$) \text{ (where M=effort/day is constant)}
- Cost(Rs) = Cost/unit effort (\text{Rs}) * Total Effort($\Phi$)
- Production(p) $\propto$ Effort($\Phi$)

As shown by Brooks (Broo75) this does not work for software development process because software production rates
(productivity) are not constant with time and production is not proportional to effort. Hence, mathematical calculus must be used for this purpose.

ILC Methodology is based on Norden's work (Fig. 5.3) according to which there are regular patterns of manpower build-up and phase-out in projects. Each phase of a project shows growth and decay patterns in manpower effort across time.

5.5 Integrated Life Cycle Equation and Project Curves

As developed in chapter 4, set (4.14) is the normalised form of life cycle equation. By introducing a parameter $\Phi$ expressed in terms of total Integrated Life Cycle effort we can rewrite the integral and derivative forms of ILC equation as follows:

Cumulative Effort $\Phi(t) = \Phi(1 - e^{-\alpha t^2/2})$ ... (5.1)
Effort/unit time $\dot{\Phi}(t) = \Phi \alpha t e^{-\alpha t^2/2}$ ... (5.2)

The Rayleigh curve closely fits the manpower effort in each phase of a homogenous project. When the overlapping curves are added together they result in a project curve of pure Rayleigh shape (Fig. 5.6).

The project curve shows that a software project does not start full-blown. Actually, the manpower which is proportional to effort, builds up at a rate determined by shape parameter $\alpha$. Sharply peaked manpower build-up corresponds to a 'crash' project while relatively flat curve is associated with relaxed or 'stretched-out' projects (Fig. 5.7). This should not exceed 30% otherwise it may strain the informal project structure.
On the project curve there are two significant points 'peak' and 'tail'. The long tail of the project curve supports the view that in case of software projects percentage completion is a meaningless concept i.e. projects remain nearly complete for a long time. From (5.1) it is evident that 90% of the work is accomplished in about two-thirds of the development time. That is why projects tend to perch at 90% complete level. In the real world, requirements are never fixed and changes to product specification occur frequently, \( \Phi \) and \( \tau \) are not completely fixed or deterministic. Also, management may not respond perfectly to cues from the project resources. So the system has random or stochastic component superimposed on the deterministic behaviour. This 'noise' is shown in Fig. 5.6 by the dotted lines.

The shape parameter \( a \) depends on the point in time at which \( \phi \) reaches its maximum i.e.

\[
a = \frac{1}{\tau^2}
\]

\[\ldots \ldots (5.3)\]

where \( \tau \) is the time to reach peak effort. In fact, \( \tau \) corresponds to the development time - time to reach initial operational capacity. Substituting \( a \) we can write ILC equations as follows:

\[
\phi = \Phi (1 - e^{-t/2\tau})
\]

\[\ldots \ldots (5.4)\]

\[
\phi = \left(\Phi / \tau\right) \cdot t \cdot e^{-t/2\tau}
\]

\[\ldots \ldots (5.5)\]

Development effort

\[
\Phi_\tau = \Phi (1 - e^{-\tau/2\tau})
\]

\[= 0 \cdot (1 - 1/\tau \epsilon)\]

\[\approx 0.45 \Phi \quad (sec. \ 5.9) \quad \ldots \ldots (5.6)\]
The greater part of software cost is proportional to the cost of the people hence ILC cost is essentially $\Phi$ times the average burdened cost/unit time called the 'standard cost' (neglecting computer time, inflation, overtime and other overheads).

Development Cost $R_{st} = 0.45 R_{ILC}$ \hspace{1cm} (5.7)

5.6 **Empirical Evidence**

There are obviously administrative problems in collecting accurate manpower data for projects. Norden (Nord70) applied life cycle curves to about 20 engineering projects with encouraging success. Putnam (Put77) applied this to software projects with reservations (Fig. 5.4). We have applied this to all the seven phases of software life cycle of a project (Fig. 5.5).

$\Phi$ is the total manpower expended when a project is initiated, the proposed budget is an estimate of $\Phi$ and the available manpower permits $\alpha$ to be calculated. Assuming that requirements analysis determines that these figures represent an accurate assessment of the complexity of the problem, the estimated development data may be computed and cannot be set arbitrarily during the requirements analysis/specification phase.

If we have a reasonable estimate of $\Phi$, we can determine $\alpha$ by estimating initial slope of $\phi'$. In general, we can use regression analysis to determine $\Phi$ and $\alpha$ from data. We have applied this technique (Fig. 5.8 ) to Safeguard Project data (Step76).
5.7 Linear ILC Equation

Dividing both sides of (5.5) by \( t \) and taking \( \log_{10} \) and redefining terms:

\[
\log \left( \frac{\phi}{t} \right) = \left( -\log e / 2t^2 \right) t^2 + \log(\phi/t^2) \quad \ldots (5.8)
\]

This equation is of the form \( y = mx + c \) where

\[
y = \log \left( \frac{\phi}{t} \right) \quad m = - \log e / 2t^2 \quad c = \log \left( \frac{\phi}{t^2} \right) \quad \ldots (5.9)
\]

If we plot \( y \) versus \( t^2 \) on log-log graph we get a straight line where \( m \) and \( c \) are slope and intercept as shown in Fig. 5.9.

By intuition, one would suppose larger the project (\( \phi \)), harder it will be to accomplish. Increasing the project size involves more levels of management, greater problems of co-ordination and geometrically increasing communication problems (GuTo84). It also seems reasonable that shorter the development time (\( t \)), greater is the difficulty. More effort has to be packed in less time. Therefore, it seems likely that complexity is directly proportional to project size and inversely proportional to the development time.

In fact, the factor \( 8 = \frac{\phi}{t^2} \) argument of the intercept in Fig. 5.9 is a measure of complexity. In dimensional terms it expresses the rate of change in effort expended. \( 8 \) is a constant for a given project. The projects may be classified according to the value of \( 8 \) as follows:

<table>
<thead>
<tr>
<th>S.No</th>
<th>Project Type</th>
<th>Complexity(8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Very Easy</td>
<td>1-9</td>
</tr>
<tr>
<td>2.</td>
<td>Easy</td>
<td>10-29</td>
</tr>
<tr>
<td>3.</td>
<td>Medium</td>
<td>30-99</td>
</tr>
<tr>
<td>4.</td>
<td>Hard</td>
<td>100-149</td>
</tr>
<tr>
<td>5.</td>
<td>Very Hard</td>
<td>( \geq 150 )</td>
</tr>
</tbody>
</table>
5.8 **Software Complexity, Manageability and Dexterity**

When the complexity \((8)\) is plotted against productivity \((p)\) (tested and documented code production rate) on a log-log graph for a number of systems we find a good correlation as shown in Fig. 5.10a.

Additional evidence indicate that the situation is more interesting and that there is a family of parallel lines shown in Fig. 5.10b and given by:

\[
p = T_n 8^{-2/3} \quad \text{or} \quad p 8^{2/3} = T_n \quad \ldots \ldots \quad (5.10)
\]

The constants are regularly spaced and are given by Fibonacci sequence \(T_{n+2} = T_n + T_{n+1} \quad \ldots \ldots \quad (5.11)\)

These are interpreted to be measure of state of technology being applied to project or the software engineering environment in which project is being executed. This SEE constant \(T\) depends on the following factors (GuTo84):

1. Throughput of the SEE
2. Mode of development - high for online interactive mode and low for batch mode.
3. Design methodology adopted.
4. Modern Programming Practices (MPP)
5. Usage of language level etc.

5.8.1 **Iso-tech Lines**

(5.10) indicates that there is close analogy with the well-known Gas Law \((\text{PV}^\gamma = \text{constant})\). We can define Iso-tech lines (analogous to isobars or isothermal lines) for various software houses as shown in Fig. 5.10b. For a particular
computer configuration the software development environment tends to fall on a particular line. With improved systems the tendency is to move upwards as shown in Fig 5.10b.

5.8.2 The Project Manageability depends on the interaction of three parameters - complexity $\Phi$, development time $\tau$ and project size $\Phi$. In practice, the range of these parameters is not indefinite. The time region is constrained at the minimum end by the manpower build-up rate $\alpha$ which is possible and at the other end by economic considerations. There is a development time beyond which a product is no longer of interest. When the feasible region is portrayed in 3D by introducing complexity $\Phi = \Phi/\tau^2$ as a third dimension, it can be represented as a complexity surface (Fig. 5.11).

It will be seen that complexity increases dramatically as the development time is shortened. Systems tend to fall on a set of lines which are the trace of a constant magnitude of complexity gradient. Each line is a characteristic of expertise to do a certain class of work.

5.8.3 The rate of change of complexity can be studied by taking the gradient of $\Phi$. The unit vector $\mathbf{i}$ points in $\tau$ direction. The unit vector $\mathbf{j}$ points in $\Phi$ direction. Grad $\Phi$ points almost completely in $\mathbf{-i/-\tau}$ direction.

The significance of this fact is that shortening development time by management, dramatically increases the project complexity, usually to an impossible level.

Mathematically $\nabla \Phi = -\Phi/\tau^3 \mathbf{i} + \frac{1}{\tau^2} \mathbf{j}$ \hspace{1cm} \cdots \cdots \hspace{1cm}(5.12)$

The magnitude of the $\mathbf{i}$ component is $\Phi/\tau^3$. This is the
major component and points in the negative \( \tau \) direction. The trace of all such points is a line \( \Phi = n \tau^3 \). Such a line appears to be a measure of dexeterity (skill or expertise) of project personnel to solve problems and decision making and seems to be the maximum complexity gradient that a software house is capable of accomplishing. As the system size is increased, the development time will also increase so as to remain on a line of constant magnitude of gradient defined by \( \eta = \frac{\Phi}{\tau^3} \) (Le Chatlier's Principle). The values of dexeterity constant \( \eta \) for various systems is found to be as follows:

1. New product with interfacing with old product \( \eta = 10 \) (most complex);
2. New stand-alone product \( \eta = 15 \) (less complex);
3. Rebuild for existing product \( \eta = 30 \) (least complex).

5.9 The ILC Throughput Equation

In order to derive a relationship between \( \Phi, \tau \) and \( \Phi \)
documented and tested code - programming throughput) we proceed as follows. The individual equation of the curve for each phase may be written as:

- Requirements Analysis: \( \phi_1 = \left( \frac{\Phi_1}{\tau^2} \right) t e^{-\frac{\tau^2}{2}} \)
- Specifications: \( \phi_2 = \left( \frac{\Phi_2}{\tau^2} \right) t e^{-\frac{\tau^2}{2}} \)
- Design: \( \phi_3 = \left( \frac{\Phi_3}{\tau^2} \right) t e^{-\frac{\tau^2}{2}} \)
- Coding: \( \phi_4 = \left( \frac{\Phi_4}{\tau^2} \right) t e^{-\frac{\tau^2}{2}} \)
- Testing: \( \phi_5 = \left( \frac{\Phi_5}{\tau^2} \right) t e^{-\frac{\tau^2}{2}} \)
- Documentation: \( \phi_6 = \left( \frac{\Phi_6}{\tau^2} \right) t e^{-\frac{\tau^2}{2}} \)
- Operation & Maint.: \( \phi_7 = \left( \frac{\Phi_7}{\tau^2} \right) t e^{-\frac{\tau^2}{2}} \)

Since the project criteria of homogeniety and
connectedness are satisfied, we can sum up these curves:

\[
\Phi_t \text{ is given by } \sum_{i=1}^{6} \phi_i = \sum_{i=1}^{6} \left( \frac{\phi_i}{t_i} \right) t_i e^{-t^2/2t_i^2}
\]

\[
\Phi_{ILC} \text{ is given by } \sum_{i=1}^{7} \phi_i = \sum_{i=1}^{7} \left( \frac{\phi_i}{t_i} \right) t_i e^{-t^2/2t_i^2}
\]

\[
\Phi_t = 0.45 \Phi_{ILC} \quad \ldots \ldots (5.14)
\]

From empirical evidence development effort is found to be 45% of ILC effort (Fig. 5.5).

Let \( \dot{E} \) = rate of documented and tested code

Then \[
\frac{d\dot{E}}{dt} = \dot{E} = p_0 \quad \text{(where } p \text{ is productivity)}
\]

\[
\dot{E} = \int_0^\tau \frac{d\dot{E}}{dt} \, dt = \int_0^\tau p_0 \, dt
\]

\[
= p_0 \int_0^\tau 1 \, dt
\]

\[
= p_0 (0.45 \Phi)
\]

Substituting \( p \) from (5.10)

\[
\dot{E} = T_n \left( \frac{\Phi}{\tau^2} \right)^{-2/3} (0.45\Phi)
\]

\[
= T_\phi \Phi^{1/3} \tau^{4/3}
\]

Hence \[
\dot{E} \propto \Phi^{1/3}
\]

\[
\dot{E} \propto \tau^{4/3}
\]

The ILC Software Throughput equation can now be written as:

\[
\dot{E} = T_\phi \left( \frac{\Phi^4}{\tau^4} \right)^{1/3} \quad \ldots \ldots \quad (5.15)
\]

5.10 The Effort-Time Trade-off Relation

(5.15) may be transformed into productivity expression as follows:

\[
p = \frac{\dot{E}}{\Phi_t}
\]

\[
= T_\phi \Phi^{1/3} \tau^{4/3} / \Phi_t
\]

76
\[
T = \frac{T_\phi \Phi^{1/3} \tau^{4/3}}{\Phi_t} = T_\phi \left(\frac{\Phi_t}{0.45}\right)^{1/3} \tau^{4/3} = T_\phi \left(\frac{\tau^2}{\Phi_t}\right)^{2/3} \quad \ldots \ldots (5.16)
\]

This implies that in order to increase productivity it is necessary to:

1. increase the Environment and Technology constant \((T_\phi)\)
2. increase the development time \((\tau)\)
3. decrease the development effort \((\Phi_t)\)

Also, project size implies \(\Phi_t^4 = \text{constant}\). The high power of \(\tau\) emphasizes time sensitivity of projects. The \(\tau\) is compressible to the maximum build-up rate \(\alpha\). So the effort-time trade-off relation is:

\[
\Phi_t \alpha = \frac{1}{\tau^4} \quad \ldots \ldots (5.17)
\]

### 5.11 Summary

To summarize the conclusion drawn from the investigations of this chapter we emphasize that software development exhibits characteristic behaviour represented by classical Rayleigh curve. Software development process is dynamic and not static. Programmer productivity varies and depends on Technology Constant and Software Engineering Environment under which the project is executed. It is a function of system complexity and cannot be arbitrarily increased. Software development is very much time sensitive. There is a minimum time to do a job (nine women can't produce a baby in one month if one woman takes nine!). Time and manpower are not freely interchangeable.
DEVELOP REQUIREMENTS FROM REQUEST FOR QUOTATIONS (RFQ)

COLLECT INFORMATION FROM SIMILAR PROJECTS

SELECT BASIC RELEVANT DATA

DEVELOP ESTIMATES BY
   COMPARING THIS PROJECT TO SIMILAR PROJECT
   DIVIDING THIS PROJECT INTO UNITS
   SCHEDULING WORK BY MONTH AND ESTIMATING RESOURCE / MONTH.
   DEVELOPING STANDARDS THAT CAN BE APPLIED TO WORK

ESTIMATE COMPLETE?

YES

FINAL EVALUATION

STOP

FIG. 5.1 - COMMON PRACTICE FOR PROJECT RESOURCE ESTIMATION
FIG. 5.2 - SPECTRUM OF RESOURCE MODELS
CUMULATIVE MANPOWER UTILIZATION

\[ \phi = \Phi \left( 1 - e^{-\alpha t^2/2} \right) \]

CURRENT MANPOWER UTILIZATION

\[ \dot{\phi} = \dot{\Phi} e^{-\alpha t^2/2} \]

FIG. 5.6 - MANPOWER UTILIZATION

DISTRIBUTION OF SAME TOTAL EFFORT VARYING TIME REQUIRED TO REACH TOTAL EFFORT

DISTRIBUTION WITH CONSTANT TIME BUT VARYING TOTAL EFFORT

FIG. 5.7 - MANPOWER UTILIZATION DISTRIBUTION CURVES
\[ \log(\phi/t) \]

**FIG. 5.9(a) - ILC CURVE**

\[ \log C = \log(0/c^2) \]
\[ \text{SLOPE } m = \log e/2c^2 \]

**FIG. 5.9(b) - LINEAR ILC Eq.**

\[ \eta^{2/3} = T \]
\[ PV^T = K \]

**FIG. 5.10(a) - PRODUCTIVITY Vs COMPLI**

\[ \eta = T \eta \]
\[ \text{EASY} \]
\[ \text{ISO-TECH LINES} \]
\[ T_3 \]
\[ T_2 \]
\[ T_1 \]
\[ \text{HARD} \]

**FIG. 5.8 - PROJECT SAFEGUARD MANPOWER PLOT**

**FIG. 5.10(b) - PRODUCTIVITY Vs COMPLEX**

(LOG-LOG PL)