Software Project Planning & Control with System Dynamics & Neural Networks

"When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot....your knowledge is of a meagre and unsatisfactory kind"

Lord Kelvin

3.0. Introduction

It is well known that large scale system projects, once started, get surreptitiously out of hand, and it is not unusual for the project to double its cost by delivery time. For the manager, managing on quicksand has become a way of life; for the customer, getting progressively accustomed to an ever increasing budget is commonplace. It is therefore important to devise techniques to help the project manager keep the project under control.

One of the conditions for this to be possible is for the software development plan to be based on realistic estimates. The software estimating process requires the basic understanding that software development is not a "mechanistic" process, in that the tasks are not all visible or measurable as in the case of a deterministic quantity. Software development is a "probabilistic" process, comprising a large number of tasks with strongly coupled interactions of considerable complexity. These tasks are not all
capable of being objectively measured and at best, they can be assessed through group consensus of experts in the area. Apart from this, current techniques to test software are also not very reliable, leading to uncertainty in the determination of the complete satisfaction of user requirement specifications. Software cost and development time prediction will remain no better than a probabilistic estimating method until better estimating methods are made available through a better understanding of the dynamics of software behavior, factors the management can control, and factors that are limited by the process itself.

A very strong case for effective modelling of the dynamics inherent in the system development methodology is thus made, with an objective to provide the manager with a framework to make reasonable estimates of resource, cost and schedule. There is a need not only for proper estimation, but also for rules leading to proper control and management of the project. It must be understood that the estimation models not only do the job of estimating the schedule, effort, cost etc., but also provide an essential framework for software project planning and control. They provide important "management numbers" concerning the state and parameters of the project which are critical for resource allocation and help move towards a more organized software product. Software development estimation models are "information systems" providing information to a decision making unit.

3.1. Manpower Distribution Modelling

Each new software project represents a technological advance. Project planning consists of scheduling a number of activities, each of them having their own purpose, and all these activities
contribute to the development of an overall system designed to meet a set of objective requirements.

The effort distribution for a large scale project generally takes on a classic shape first described by Lord Rayleigh as shown in Fig. 3.0(a). Norden later studied these curves and substantiated them with empirical data collected from a number of system development projects, as a result of which they are often referred to as Rayleigh-Norden curves. These curves for manpower distribution are shown in Fig.3.0(b) and are described by the equation:

\[ m(t) = 2Kae^{-at^2} \]  

where \( m(t) \) is the manpower in man-years per year (MY/YR), \( a \) is a positive number, \( K \) is the total Generic life-cycle effort in man-years (MY), and \( t \) is the project duration in years (YR).

The manpower distributions of large software projects have generally been found to follow a Rayleigh-Norden profile upon which is superimposed sufficient "noise", the latter being present within the system for a variety of reasons such as "inadequate or imprecise specifications, changes to requirements, imperfect communication within the human chain, and lack of understanding by the management of how the system behaves" [PUT78].

---

22 Development projects in this context have been defined by Norden as "...a finite sequence of purposeful, temporally ordered activities, operating on a homogeneous set of problem elements, to meet a specified set of objectives representing an increment of technological advance" [NOR50].

23 Norden makes the following four assumptions about problem-solving activity in the derivation of this model:
- The number of problems is known and finite;
- Problems are detected, recognized, analyzed and solved by means of the creative effort of the manpower involved;
- Every problem solving effort made as a result of planning or design decisions is an event that removes an unsolved problem from a finite list of problems;
- The number of people working on the project at any instant of time is proportional to the number of unsolved problems ready for solution at the same time.
FIG. 3.0 (a) CLASSIC RAYLEIGH CURVE

FIG. 3.0 (b) MANPOWER VS TIME PLOT

- Manpower MY/YR

- Generic Project Lifecycle

- TIME(YR) t_d

FIG. 3.0 (c) CUMULATIVE EFFORT VS TIME PLOT

- Cumulative effort MY

- Cumulative effort 0.4K

- TIME(YR) t_d

FIG. 3.0 THE CURVES DESCRIBED BY EQUATION 3.1, ORIGINALLY APPLIED BY LORD RAYLEIGH TO DESCRIBE OTHER SCIENTIFIC PHENOMENA, HAVE BEEN FOUND TO FIT THE MANPOWER DISTRIBUTION PATTERN OF SOFTWARE DEVELOPMENT REASONABLY WELL, AT LEAST WITHIN THE "NOISE" OF THE DATA POINTS.
Based on an extensive historical software project database comprising projects ranging from 30 MY to 1000 MY, Putnam has noticed that in large-scale software development the manpower distribution reaches its maximum very near the delivery time $t_d$. Before this time, effort is spent on specification, design, coding, testing and qualification, while after it the manpower costs correspond to maintenance, modifications and other on-site work. On the basis of empirical data it has been well established that in the Rayleigh-Norden model,

$$a = \frac{1}{2} \frac{t^2}{t_d^2} \quad \ldots (3.2)$$

which results in:

$$m(t) = K \frac{t}{t_d^2} e^{-\frac{t^2}{2t_d^2}} \quad \ldots (3.3)$$

During the course of a software project, the cumulative cost\(^24\) (which can be found by integrating equation (3.3)), increases from zero, following an S-shaped curve of the form:

$$C(t) = K (1 - e^{-\frac{t^2}{2t_d^2}}) \quad \ldots (3.4)$$

From Fig.3.0(c) it can be seen that the manpower cost at delivery time is 39% of the total cost expended on the project, with the remaining cost being directed towards on-site maintenance and modification. At $t_d$ years the cumulative manpower growth rate reaches a maximum called the peak manning $m_o$:

---

\(^{24}\) The cost of a project is measured in terms of the effort expended in man-years.

(97)
\[ m_0 = \frac{K}{t_d \sqrt{c}} \]  

...(3.5)

The slope of the manpower distribution equation at time \( t = 0 \) is defined as the Difficulty, \( D \), of the project

\[ D = \frac{K}{t_d^2} \]  

...(3.6)

which is measured in persons per year\(^{25}\).

Putnam's observations [PUT89] [LON87] suggest that if the project scale is increased, the development time also increases such that the quantity \( D_0 = (K/t_d^3) \) clusters around six different values\(^{25} - 8, 15, 27, 55, 89, \) and 233. \( D_0 \) is referred to as the manpower buildup parameter and is proportional to the time derivative of the difficulty, \( D \), of the project. We thus have the manpower buildup parameter (MB parameter) defined as:

\[ \text{MB parameter} = D_0 - \frac{K}{t_d^3} \]  

...(3.7)

where the effort \( K \) is in MY and \( t_d \) is in years.

The manpower buildup parameter varies slightly from one organization to another depending on the average skill of the analysts, programmers, and management involved. \( D_0 \) also has a strong influence on the shape of the manpower distribution - the larger its value, the steeper the curve and the manpower buildup. As \( D_0 \) is empirically defined in accordance with the nature of the software system under development, it seems reasonable during the estimating process to

\[ \text{The effort and time derivatives of the Difficulty, } D, \text{ indicate that a given software development is essentially time sensitive.} \]

\[ \text{Low for entirely new software with many interfaces and interactions with other systems, medium for new stand-alone systems and high if the software is re-built from existing reusable code.} \]

(98)
use its maximum value which provides the minimum development time, \( t_{d}^{\text{min}} \).

The manpower buildup parameter is thus a key management factor. It is a measure of the staffing style of the organization in question and is dependent on a number of factors which include the:

- task concurrency of a software project;
- schedule pressures for delivery; and
- complexity of the application in question.

The MB parameter can be translated into a data-determined scaled "management number" called the manpower buildup index (MBI) using the translation Table III.1. Various MBI levels indicating different staffing profiles are shown in Fig.3.1. Level 1 is a low, slow staff buildup, typical of sequential task execution often caused by much fundamental new design of the system, algorithms, and logic. Level 1 also takes the longest and costs the least. Level six is often referred to as a "mongolian horde" staffing style, characterized by completely

<table>
<thead>
<tr>
<th>MB parameter</th>
<th>MBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3</td>
<td>1</td>
</tr>
<tr>
<td>14.7</td>
<td>2</td>
</tr>
<tr>
<td>26.9</td>
<td>3</td>
</tr>
<tr>
<td>55.0</td>
<td>4</td>
</tr>
<tr>
<td>89.0</td>
<td>5</td>
</tr>
<tr>
<td>233.0</td>
<td>6</td>
</tr>
</tbody>
</table>
FIG. 3.1 MBTI ILLUSTRATION

FIG. 3.2 PROJECT LIFE-CYCLE
parallel task execution, with resource estimates known well in advance. Level 6 is the fastest and the most expensive.\footnote{The manpower buildup index has considerable economic implications, and thus, the final selection of the MBI for a specific project is a function of the resource constraints within the organization. For example, increasing the MBI from 1 to 3 in an effort to compress the schedule would more than double the total cost of the project. The reason for this is that the number of human communication paths for an MBI of 3 is about six times higher than that for an MBI of 1, something that would eventually manifest itself directly in terms of exponentially more defects, and therefore considerably reduced quality. Clearly then, schedule compression is an expensive proposition.}

3.2. The Generic Life-cycle and its Sub-cycles

Our discussion so far has been restricted to the Generic project life-cycle, where the manning is assumed to follow a Rayleigh/Norden distribution, with the peak manning occurring at the development time $t_d$, where the product reaches its full operational capability (FOC).

In fact, a survey of the skills of the people involved in the project indicates that the Generic life-cycle can be thought of as being comprised of a number of sub-cycles as shown in Fig.3.2, which correspond to product definition and specification; design, code and development; test and validation; adaptation and maintenance; and overall project management.

Two groups of tasks which are an integral part of the project need to be distinguished at this stage. These are: development tasks and support tasks. Development tasks include subsystems and module design, code, and test. The Development sub-cycle is the sub-cycle that deals directly with tasks concerning the production of software by such specialists as programmers, analysts, and their direct supervisors. It excludes any project-related overheads, and is assumed to follow a Rayleigh/Norden cycle itself.
The second group of tasks (support tasks) is not as clearly defined, as the first one. Into this classification fall the clerical staff, engineers that maintain the host processors and provide specific assistance to hardware usage, planning officers who run the program evaluation and review technique/critical path method (PERT/CPM) system, and configuration and control officers that keep track of the product structure of the software system and its modification.

The manpower expended in such activities, when added to the manpower consumption of the Development sub-cycle, is also assumed to follow a Rayleigh/Norden cycle delineated as the Project sub-cycle.

Software projects can be categorized on the basis of their size in source lines of code (SLOC) into three generic categories:

- small-scale projects defined by a size less than 18,000 SLOC\(^{28}\);
- Medium scale projects defined by a size between 18,000 SLOC and 70,000 SLOC\(^{29}\);
- Large scale projects with greater than 70,000 SLOC\(^{30}\).

As the system size increases, at a system size of about 18000 SLOC, the Project sub-cycle breaks away from the Development sub-cycle (due to increasing project overheads), and tends to move towards the Generic life-cycle, with which it merges around a system size of

\(^{28}\) In such projects, the overhead effort (non-technically oriented) is assumed to be almost negligible. In cases where it exists, it is incorporated into the development effort, as a result of which, the Project sub-cycle coincides with the Development sub-cycle.

\(^{29}\) In medium scale projects, the greater the size, the greater the non-technical support required to develop the product. Indeed, more effort is expended in management, configuration control, documentation, and quality assurance, and this extra effort is required to be added to the development effort. This effort grows with the size, and is added to the normal development effort. The total manpower distribution is thus represented by a distinct Project sub-cycle, and a Development sub-cycle.

\(^{30}\) Naturally, large scale projects are comprised of development effort represented by a development sub-cycle, on which is superimposed the overhead, resulting in the Project sub-cycle. Generally speaking, for large scale projects, the Project sub-cycle, tends to merge with the Generic life-cycle for software development.
70000 SLOC. Also, with this increase in system size the following points are noticeable:

- all three curves (the Development sub-cycle, the Project sub-cycle, and the Generic life-cycle) increase in size;
- the ratio between the Generic and Development cycle curves remains the same; and
- the Project sub-cycle curve first increases at a faster rate than the Generic life-cycle curve, but later on approaches the Generic curve.

Following these considerations, let $m_d(t)$ and $C_d(t)$ be the Development sub-cycle manning, and cumulative Development sub-cycle manpower cost respectively. We may thus write:

$$m_d(t) = \frac{K_d}{t_{0d}^2} \left( -\frac{t^2}{2} \right) + D_d t \left( -\frac{t^2}{2} \right)$$

(3.8)

and

$$C_d(t) = K_d \left( 1 - e^{-\frac{t^2}{2 \omega}} \right)$$

(3.9)

where $t_{0d}$ is the time at which the Development curve exhibits its peak, $D_d$ is the difficulty of the Development sub-cycle, and $K_d$ is the total effort expended in the Development sub-cycle of the system.

It is a good practical assumption that the development will be 95% complete by the time $t_d$. This gives

(103)
\[
\frac{C_d(t_d)}{K_d} = 1 - e^{-\frac{t_d}{t_{d0}}} = 0.95 \quad \ldots (3.10)
\]

and so

\[
t_{d0} = \frac{t_d}{\sqrt{6}} \quad \ldots (3.11)
\]

(all times expressed in years).

To complete the definition of the Development sub-cycle, a relationship between \(K_d\) (the total manpower effort of the Development sub-cycle) and \(K\) (the total manpower effort of the Generic life-cycle) must be established. It is quite evident that the initial slopes of the Development sub-cycle and the Generic life-cycle are the same, i.e.,:

\[
\dot{m}(0) = D - \frac{K}{t_d^2} - \frac{K_d}{t_{d0}^2} - \dot{m}_d(0) - D_d \quad \ldots (3.12)
\]

(implying that both the Generic cycle and the Development sub-cycle have the same Difficulty), and therefore

\[
K_d = \frac{K}{\sqrt{6}} \quad \ldots (3.13)
\]

The Manpower Buildup, \(D_0\), becomes:

\[
D_0 = \frac{K}{t_d^3} - \frac{K_d}{t_{d0}^3 \sqrt{6}} \quad \ldots (3.14)
\]

As mentioned earlier, the ratio of \(K\) to \(K_d\) remains the same as the system size is increased. On the other hand, the Project sub-cycle exhibits a different kind of behavior. The Project sub-cycle effort, \(K_p\), is a strong function of the system size. To study the behavior
of this sub-cycle further one can define a ratio, b, (also sometimes referred to as the special skills factor) as:

$$b = \frac{C_p(t_d)}{K} \quad \ldots (3.15)$$

where $C_p(t_d)$ is the Project sub-cycle effort expended till the development time $t_d$. In fact, it can be shown that $b$ changes with system size as shown in Table III.2.

Let us assume,

$$K_p = \frac{K}{\alpha^2} \quad \ldots (3.16)$$

where $\alpha$ represents the form factor. We can easily notice that $\alpha = \sqrt{6}$ for small scale systems and $\alpha = 1$ for large scale systems. This provides us with an interpretation consistent with the fact that for small scale systems the Project sub-cycle merges with the Development sub-cycle and we have $K_p = K/6$ ($\alpha = \sqrt{6}$), and for large systems the Project sub-cycle merges with the Generic life-cycle and we have $K_p = K$ ($\alpha = 1$). These equations provide us with a means to calculate the Project sub-cycle variables, if required, as a function of the system size. Fig. 3.3 portrays the behavior of the different cycles discussed above as a function of the system size, where $m_0$ and $t_d$ are the peak manning and peak time of the Generic life-cycle, $m_{0d}$ and $t_{0d}$ are the peak manning and peak time of the Development sub-cycle, and $m_{0p}$ and $t_{0p}$ are the peak manning and peak time of the Project sub-cycle.
**Small-Scale Project**
- \( s = 10,000 \text{ NCSS} \)
- \( t_d = 1.25 \text{ years} \)
- \( t_o = 0.5 \text{ years} \)
- \( \sqrt{6} = 0.16 \)

**Medium-Scale Project**
- \( s = 25,000 \text{ NCSS} \)
- \( t_d = 1.85 \text{ years} \)
- \( t_o = 0.76 \text{ year} \)
- \( t_p = 1.0 \text{ year} \)

**Medium-Scale Project**
- \( s = 55,000 \text{ NCSS} \)
- \( t_d = 2.6 \text{ years} \)
- \( t_o = 1.1 \text{ year} \)
- \( t_p = 2.4 \text{ year} \)

**Large-Scale Project**
- \( s = 90,000 \text{ NCSS} \)
- \( t_d = 3.2 \text{ years} \)
- \( t_o = 3.2 \text{ year} \)
- \( t_p = 1.3 \text{ year} \)

*Environmental Factor (discussed ahead)*

**NCSS**: Non-commentary source statements

**FIG. 33** REPRESENTATION OF VARIOUS PROJECT SCALES. [LON 87]
3.3. Organizational Productivity and The Software Equation

Putnam [PUT78] [PUT89] [LON87] also found that an empirical relationship exists between the overall productivity of software projects, and their respective difficulty\(^3\). It was found that the projects were grouped along three parallel lines on a logarithmic plot of the productivity versus the difficulty. Also, projects run in the same environment were grouped along the same line. After conducting a statistical analysis on the data, this fundamental behavior was formalized by the equation:

\[ Pr = C_d D^{\frac{2}{3}} \] \hspace{1cm} (3.17)

where \( D \) is the difficulty, \( C_d \) is a state of technology proportionality constant and \( Pr \) is the productivity defined by:

\[
\frac{\text{size of delivered source code}}{\text{total manpower required to produce code}}
\] \hspace{1cm} (3.18)

In the expression for productivity, the total effort to be considered is that which is expended upto the time \( t_d \), i.e., 39% of the total life-cycle effort, \( K \). Therefore we have:

\[
SS = Pr \times (0.39 \ K)
\] \hspace{1cm} (3.19)

\(^3\) Data for the study was collected from the US Army Computer Systems Command projects.
or

\[ SS - 0.39 C_a D^{\frac{2}{3}} K \] ....(3.20)

which leads to

\[ SS - 0.39 C_a K \left( \frac{K}{t_d^2} \right)^{\frac{2}{3}} \] ....(3.21)

If \( 0.39 C_a \) is replaced by a coefficient \( E \) which is the technology environment factor, or the organizational productivity parameter, then the software equation becomes:

\[ SS - 0.39 C_a K^3 t_d^{\frac{4}{3}} \] ....(3.22)

The software equation is a relation that links the functionality that has to be generated, to the time and resource expenditure (which are "management variables") required to create it. Moreover, it provides a way to measure past performance when rearranged into the following computational form:

\[ SS - E K^3 t_d^{\frac{4}{3}} \] ....(3.23)
For completed projects, the terms on the right hand side of the equation are known. Therefore, the software equation can be used to actually measure the productivity and realistically tune estimating models with real data.

The productivity parameter defined in the software equation above embraces many factors of the software development process such as:

- the influence of management;
- methods used;
- tools, techniques, and aids used;
- skills and experience of team members;
- machine service; and
- complexity of application type.

The productivity parameter for systems on the Putnam database has been found to exhibit an exponential behavior. This is not surprising given the very broad variability in time, effort, and errors. The range of values spans from a few hundred to hundreds of thousands, clustering around certain discrete values that follow a Fibonacci-like sequence. Because these numbers are not well understood by commercial managers a simple integer scale is generally used to represent them, and these translated numbers are referred to as the productivity index (PI) of the organization. Table III.3 shows this family of numbers. Values of PI from 1 to 25 are adequate to span the universe of all software projects seen so far, and Table III.3 can be extended whenever an organization becomes efficient enough to require it. It can be seen that not much
information is required to calculate the PI: the total new and modified source lines of code; the total man-years of effort; and the total elapsed calendar years spent on a project, are the only pieces of information required.

The PI is a macro-measure of the total development environment. Low values of PI are associated with low productivity environments, poor tools or high complexity of software. High values are associated with good environments, tools and management, as well as well understood projects of low complexity. Average PI values range from 2-3 for complex applications such as microcode design and real time embedded systems, to 12-15 for scientific and business application systems.

The economic leverage of an organizational productivity improvement is significant because the PI represents an exponential family.³²

<table>
<thead>
<tr>
<th>Productivity Parameter</th>
<th>PI</th>
</tr>
</thead>
<tbody>
<tr>
<td>754</td>
<td>1</td>
</tr>
<tr>
<td>987</td>
<td>2</td>
</tr>
<tr>
<td>1220</td>
<td>3</td>
</tr>
<tr>
<td>1597</td>
<td>4</td>
</tr>
<tr>
<td>1974</td>
<td>5</td>
</tr>
<tr>
<td>2584</td>
<td>6</td>
</tr>
<tr>
<td>3194</td>
<td>7</td>
</tr>
<tr>
<td>4181</td>
<td>8</td>
</tr>
<tr>
<td>5186</td>
<td>9</td>
</tr>
<tr>
<td>6765</td>
<td>10</td>
</tr>
<tr>
<td>8362</td>
<td>11</td>
</tr>
<tr>
<td>10946</td>
<td>12</td>
</tr>
<tr>
<td>13530</td>
<td>13</td>
</tr>
<tr>
<td>17711</td>
<td>14</td>
</tr>
<tr>
<td>21892</td>
<td>15</td>
</tr>
<tr>
<td>28657</td>
<td>16</td>
</tr>
<tr>
<td>35422</td>
<td>17</td>
</tr>
<tr>
<td>46368</td>
<td>18</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

³² A productivity improvement is producing an equal or better quality product with fewer people, in less time, for less money, a definition which can be immediately interpreted in terms of the resources required: if an organization exhibits a higher productivity, the resources required are smaller, and vice-versa. A capital investment on an organization's environmental bottleneck could cause productivity jumps of 2 or 3. An increase in the PI by one for a 30,000 SLOC COBOL system saves close to US$ 0.25 million. This means that the PI is the capital investment term in software development. Also, with increases in PI, time, effort, manpower, cost and defects, all decrease.
To summarize, an increase in the PI of an organization:

- reduces the time of development;
- reduces the effort;
- reduces the defects;
- increases the mean time to failure (MTTF);
- reduces the manpower requirement;
- increases the code production rate (SLOC/MONTH);
- increases the code/man-month ratio (SLOC/MAN-MONTH).

Fig. 3.4 shows a parametric graph of the software equation. The darker line labelled CONSTRAINT represents the minimum time in which a system can be developed. The area below the line represents the region in which it is not feasible to attempt the development of a software system. For example, in the graph we can see that it is not feasible to develop a system of 200,000 SLOC in less than 2.45 years, leading to the concept of minimum development time which is discussed ahead in more detail. There are other constraint conditions depending upon the type of system. This particular constraint applies to a stand alone system that must be designed and coded from scratch. While this graph shows the functional relationship between size, effort and time for all systems, absolute values will differ for different organizations, and even for projects within an organization.

3.4. Software Sizing

During the early systems definition phase one needs a broad estimate of the expected system size. At this point of time no hard data regarding the system is available, as no design has been done. Therefore, all that can be really done is to make an intelligent guess about the range of the size of the system based on past
FIG. 3.4 SIZE - EFFORT - TIME TRADE OFF - CHART

E = 10040  Constraint = 14.7 (Do)

Effort values in this graph relate to the project sub-cycle effort: Cp(td) = 0.4 K for large systems has been assumed.
knowledge. If 'a' is the lowest possible number of source statements estimated, and 'b' the highest number, one can determine the expected size and standard deviation of the system using the laws of statistics and probability.

To start with, the system is divided into as many different functional blocks as can be estimated at the commencement of the project. For each of these functional modules the smallest and largest source code sizes are estimated. This may be done typically using a Delphi polling questionnaire. The expected number of source statements for the $i^{th}$ module is then

$$SS(i) = \left( \frac{a + b}{2} \right)$$ \hspace{1cm} \ldots (3.25)$$

and the standard deviation,

$$\sigma_{SS(i)} = \frac{|b - a|}{6}$$ \hspace{1cm} \ldots (3.26)$$

Using these estimates of sizes and standard deviations of the different functional modules, an initial assessment of the overall expected size and standard deviation for the system may be made.

A little later in the life-cycle, around the beginning of functional design, a better estimate of the size of components should be possible as the functional blocks are clearer. Once again the smallest, largest, and also the most likely size, $m$, of different functional blocks are estimated. Expected values are then calculated assuming a skewed distribution:
which simply biases the result so that the expected value falls on the side about which we are more uncertain; and standard deviation

$$
\sigma_{SS(i)} = \frac{|b - a|}{6}
$$

....(3.28)

The overall expected size is the sum of the expected sizes of the individual modules, and the standard deviation is simply the square root of the sum of the squares of the individual standard deviations.

The shape of the curve obtained by analyzing data acquired from the Delphi poll, would actually indicate the degree of certainty and risk in the estimates obtained. As shown in Fig.3.5, components of the software system that are to be counted for the size planning approach for rebuilt software systems will include: new components, modified components, and components that are unmodified but require testing or reworking. Using this data, along with data on standard components (input-output screens, subsystem modules), function providing elements, knowledge about database applications, one can get rather reliable estimates for the size of the software and associated uncertainty using Bayesian statistical weight techniques as shown in Fig.3.6.
FIG. 3.5 RE-USED SOURCE CODE CASE

FIG. 3.6 SIZE ESTIMATION PROCESS

FIG. 3.7 SOFTWARE EQUATION
3.5. The Minimum Time Concept

It is extremely important for software managers to understand that there is a minimum time required to complete any software development. This minimum time, is a function of:

- the number of lines of code required to implement the functionality of the system;
- the inherent complexity of the application type;
- the efficiency of the development group; and
- the manpower buildup rate.

If one re-arranges the software equation one can plot the log of K and log of $t_d$ as shown in Fig.3.7. The software equation is defined by an estimate of the number of lines of code required to implement the system and the measured efficiency of the development organization (PI). Similarly, the MBI equation can also be plotted (once again on a log K - log $t_d$ scale) as shown in Fig.3.8. The MBI defines the maximum effective application of manpower for a given design complexity. Solving both these equations simultaneously (Fig.3.9) for a given project size and a PI representative of an organization's capability, the region to the left of these lines represents the infeasible region. Hence, there is a minimum time, $t_d^{[\text{in}]}$, which also represents the maximum cost solution for the system, requiring the most people. It should be noted that $t_d^{[\text{in}]}$ is obtained for the maximum recommended value of the MBI for the application in question, whereas $t_d$ could represent the delivery time for a more relaxed value of the MBI which could have been selected in order to avoid a violation of one or more of the constraints within the organization.
The manpower buildup index defines the maximum effective application of manpower for a given design complexity.

**FIG.3.8 MBI EQUATION PLOT**

Feasible region for development

**FIG.3.9 SIMULTANEOUS SOLUTION OF SOFTWARE EQUATION AND MBI EQUATION**

**FIG.3.10 CONSTRUCTION OF THE PLANNING ZONE**
3.6. The Putnam Modelling Methodology: Linear Programming Technique and The Planning Zone

The fundamental goal of the Putnam Modelling methodology is the construction of a planning zone from where the project manager can construct a manning profile and develop a project plan.

Construction of a planning zone is preceded by a size estimate (or nominal size range) which is typically dependent upon the initial statement of user requirements. The case for re-usable code (deleted, changed, added and modified) has to be taken into account at this stage. This code may be code which is derived from modules used in an unmodified form, or which are slightly modified, in which case they would comprise some changed code and might have some added code after deleting some code.

An estimate of the technology environmental factor, $E$, is made from similar past projects of the same organization. In the absence of a historical database for the organization concerned, similar software projects developed in a similar environment of an equivalent organization are used as a reference for the first estimate. Typically, logarithmic plots of the ratio, $Pr$, of the size of the delivered source code to the total manpower effort required to produce the code up to the time $t_d$ ($= 0.39 K$), and the difficulty, $D$ ($= K/t_d^2$), are used to find a line of best fit, from which the technology environmental factor, $E$ (also known as the organizational productivity parameter), of the organization can be estimated accurately.

(118)
As has already been discussed, the manpower buildup is decided depending upon the application in question, by choosing values from those given by Putnam (Table III.1)\textsuperscript{33}.

The planning zone is then built up from the following five constraints as shown in Fig.3.10 by the area ABCDE:

- **cost constraints** - the maximum cost ($C_{\text{max}}$) above which the organization cannot afford to go, and the minimum cost ($C_{\text{min}}$) below which the organization does not wish to be involved because of possibly poor return on capital;

- **time constraints** - the maximum time ($t_{\text{max}}$) is limited by the delivery time which depends on the customer demand, and the earliest time ($t_{\text{min}}$) is that before which the customer may not wish to receive the product;

- **peak manning constraints** - a maximum manning ($m_{\text{max}}$) above which either the project becomes unmanageable or which resources do not allow, and the minimum manning ($m_{\text{min}}$) below which the manager cannot go because resources cannot be re-allocated;

- **manpower buildup constraints** - this is the maximum value not to go beyond ($D_0$), as has already been mentioned, and it is also practical to choose a minimum value ($d_0$) which can help define the longest development time;

- **difficulty constraints** - representative of the slope at the origin of the manpower profile (also called the team

\textsuperscript{33} The justification for using them is that these values have been validated by extensive statistical work.

(119)
growth) - a maximum difficulty (D) being defined by the cohesiveness of the team (since new people need to be properly integrated into the team), and the minimum team growth (d) being defined by the minimum number of skilled persons required to solve the problem initially, and for the manning to attain its peak at the right time.

By representing the software equation, \( S/E \), on the same curve, we can see that the line intersects the polygon ABCDE on the segment \((X_1, X_2)\). On this segment, a point \(P\) can be selected, called the planning point, which defines a workable relationship between the manpower cost of development, \(C_d\), and the time of development, \(t_d\). For risk avoidance reasons, the planning point should not be selected too close to the points \(X_1\) and \(X_2\). Fig.3.11 summarizes the essential steps followed in the Putnam modelling methodology.

As an example, consider a system of size \(SS = 98500\) SLOC with an organizational productivity parameter \(E\) of 10040, a manpower buildup parameter \(D_0 (= K/t_d^3)\) of 14.7, and a labor rate of $50,000/MY. Table III.4 shows typical constraints encountered by the manager, and exemplar solutions for the two extreme cases, minimum time, and minimum cost. The feasible development region is now identified between these two extreme solutions. In being able to invoke this powerful technique, we produce two constrained optimal solutions, the best that can be done within the constraints, and all other short-run (taking PI as given) feasible effort-time choices. A range of feasible effort-time-cost opportunities are identified in Table III.5, giving an idea of the amount of leverage available to a manager on a typical software development project.

(120)
FIG.3.11 PUTNAM BASED ESTIMATION METHOD
### Table III.4 Typical Software Managerial Constraints

**INPUTS (MANAGEMENT CONSTRAINTS)**

- Maximum number of people available at peak manloading (hiring constraint) 30 people
- Minimum skilled staff requirements 15 people
- Maximum cost $2 million
- Maximum Time 2 years

**OUTPUTS (TWO SOLUTIONS)**

<table>
<thead>
<tr>
<th></th>
<th>MINIMUM TIME</th>
<th>MINIMUM COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_0$</td>
<td>14.7 MY/Y³</td>
<td>7.3 MY/Y³</td>
</tr>
<tr>
<td>$t_d$</td>
<td>1.812 YRS</td>
<td>2.0 YRS</td>
</tr>
<tr>
<td>$K^1$</td>
<td>87.51 MY</td>
<td>58.66 MY</td>
</tr>
<tr>
<td>$C_p(t_d)$</td>
<td>34.13 MY</td>
<td>22.87 MY</td>
</tr>
<tr>
<td>$COST$</td>
<td>$1.7 M$</td>
<td>$1.14 M$</td>
</tr>
<tr>
<td>$m_0$</td>
<td>30</td>
<td>18</td>
</tr>
</tbody>
</table>

### Table III.5 Feasible Time-effort-cost solutions

<table>
<thead>
<tr>
<th>MB PARAM (MY/Y³)</th>
<th>DEV. TIME (Y)</th>
<th>$C_p(t_d)$ (MY)</th>
<th>PROJECT COST (US $ M)</th>
<th>PEAK MANNING (MEN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.7</td>
<td>1.812</td>
<td>34.13</td>
<td>1.706</td>
<td>30</td>
</tr>
<tr>
<td>12.0</td>
<td>1.865</td>
<td>30.39</td>
<td>1.519</td>
<td>26</td>
</tr>
<tr>
<td>10.0</td>
<td>1.914</td>
<td>27.38</td>
<td>1.369</td>
<td>23</td>
</tr>
<tr>
<td>8.0</td>
<td>1.976</td>
<td>24.10</td>
<td>1.205</td>
<td>19</td>
</tr>
<tr>
<td>7.3</td>
<td>2.003</td>
<td>22.87</td>
<td>1.143</td>
<td>18</td>
</tr>
</tbody>
</table>

SLOC = 98500  
$E = 10040$  
MAX MBI = 2  
LABOR RATE = 50000 $/MY
3.7. Management Trade-off Opportunities

Various management trade-off opportunities exist, which can be exploited whenever there is extreme market pressure to deliver the software product, and/or the schedule for the development of the system is unreasonably short. These trade-offs can be classified into two broad categories, short term and long term trade-offs. Short term trade-offs include product feature-function trade-off and project staffing style trade-off. Long term trade-offs encompass organizational productivity capability improvement strategies.

The product feature-function trade-off is a reduced size planning strategy which requires that the development team prioritize the functions of the system. Modules that are critical to the operation of the software may be developed first, while those that are less important be deferred to a subsequent release. The scaled down product is then delivered at the scheduled delivery date. Subsequent releases are then planned to deliver the deferred functions. For example, if a product with 12 business functions (60,000 SLOC) and an organizational PI of 15 requires 13 months minimum time for development and the marketing delivery date allows only 9 months, it would be possible to meet the delivery deadline if only 8 functions (40,000 SLOC) were implemented in the first release.

Project staffing trade-offs arise when the project staff is increased to meet tight schedule constraints, or available project staff is limited, or the project staff is limited by financial constraints, or if the project staff itself is limited by the application design complexity and problem solving cannot be done in parallel. In this trade-off, the strategy adopted is to identify all constraints (of staff, schedule, budget, complexity etc.). These constraints are then prioritized to indicate which are most
important and which are least important. A window of doable staffing plan is then determined. The effectiveness of different staffing strategies is then evaluated along with its benefits. For example, Table III.5 gives a classic example of project staffing tradeoffs within an available window. It is thus obvious that the MBI has very strong implications for management trade-off opportunities.

Organizational productivity trade-offs arise when the commercial management would like to substantially increase the software development capability to enhance corporate competitive positions, or if the technical management would like to increase the productivity in order to meet tight scheduling. Taking the example of Table III.5, if the productivity parameter of the organization were to be increased to 13530 (PI up about 1 from 12 to 13), the minimum development time would reduce to 1.59 years, Project sub-cycle effort to development time, $C_p(t_d)$ to 23.25 MY, and the peak manpower to 23 people.

Therefore, for immediate effects, the management can change project staffing levels, or modify the product feature-functionality for the first release. In the longer term, it may decide to improve the tools, methods, practices, skills and experience of its staff, all to boost the PI of the organization. There are, however, limits to the amount the management can tradeoff. For example, in the case of schedule compression, the limiting factors would be minimum time, maximum budget, and maximum available staff. In the case of schedule stretch-out, there is a minimum skilled staff requirement, and the latest acceptable delivery date. Looking into these limits, the management has a practical window within which some leverage is possible, and this bounded feasible region can sometimes lead to considerable resource savings if properly utilized.
3.8. Risk Analysis and Monte Carlo Simulation

The size planning approach adopted earlier helps calculate the degree of uncertainty in the estimates, identify areas with high risk, and estimate the management reserves required to ensure successful completion of the project on time and within the budget.

Risk planning takes into account the uncertainties in different inputs to the model to determine the amount of management reserve required to guarantee the successful completion of the project on schedule and within budget constraints. The various areas of uncertainty in the software development process are:

- the source lines of code;
- manpower buildup index;
- productivity index;
- labor rate; and
- the inflation rate.

The risk planning approach involves the use of Monte Carlo simulation, which provides a mechanism to map the uncertainties in our technical input to uncertainties in the management numbers, i.e., schedule, effort, cost, manpower, and reliability. Input variability has been summarized in Fig. 3.12.

Assuming that the input variability values are well known, the risk assessment method can be summarized as follows:

1. Estimate the range of expected values for each of the input factors, and the probability distribution for each;
2. Select at random, a single value, from the distribution of values for each factor;
(Values are well known)

\[
\begin{align*}
\text{SIZE} & : 100,000 \pm 8000 \\
\text{PI} & : 13 \\
\text{MBI} & : 2 \pm 15\% \\
\text{LABOUR RATE} & : 100,000/MY \pm 800 \\
\text{INFLATION} & : 4\% \text{ANNUAL} \pm 0.5\%
\end{align*}
\]

FIG. 3.12 INPUT VARIABILITY SUMMARY

\[\text{Log Effort} \quad \text{PI} = 13 \quad \text{SLOC} = 100,000 \pm 8000 \quad \text{MBI} 2 \pm 15\% \]

\[
\begin{align*}
287 \text{ MM} & \pm 40 \\
19.4 \text{ MOS} & \pm 0.8 \text{ MOS}
\end{align*}
\]

FIG. 3.13 MONTE CARLO SIMULATION

\[\text{Probability that cost will not exceed ceiling} \]

\[
\begin{align*}
100\% & \quad 84\% \\
50\% & \quad 34\% \\
15\% & \quad 16\% \\
0\% & \quad 247 \quad 287 \quad 327 \quad \text{Effort (MM)}
\end{align*}
\]

FIG. 3.14 CONSTRUCTING A RISK PROFILE
3. Solve the software equation and the MBI equation to get a solution for a particular combination of values;
4. Run the problem on a computer several thousand times to get a probability distribution;
5. Determine the standard deviation of the schedule, effort and cost;
6. Construct a risk profile for the schedule, effort and cost.

This procedure has been depicted in Figs. 3.13 and 3.14. Using this data, one can "design to risk". "Design to risk" implies finding a plan that will have a high probability of not exceeding a target end date. By taking the risk profile, one can plan the development process such that there is, say, for example, a 99% probability of not taking longer than the target date as shown in Fig. 3.15, where the earlier example has been shown for a 24 month delivery date.

One can also attempt joint probability planning which is finding a plan that has the highest chance of not exceeding the maximum budget and the target delivery date. This is known as "best bid" planning, where both the risk profiles of schedule and effort are taken into account while planning the development schedule. In "best bid" planning, if a plan promises a 50% chance of compliance to schedule and cost constraints, it implies a 25% joint probability of being within the planned schedule and budget. Fig. 3.16 shows an example of the best bid planning approach.

3.9. Software Reliability

A crucial factor to take into account is the reliability of the software to be developed. Software developed is seldom free from
Monte Carlo Simulation profile

Log effort (MM)

24 MONTH DELIVERY

Working plan to complete development in 21.8 MONTHS 174 MM, with 99% probability of not exceeding 24 MONTHS.

FIG. 3.15 DESIGN TO RISK (99%)

Log effort

24 MONTHS

2,000,000

97% probability of not exceeding cost and schedule.

(100,000 ± 8000, PI=13)

FIG. 3.16 BEST BID PLANNING

Defects/month

defects design
defects code & unit test defects systems test defects latent defects

Elapsed calendar time (months)

FIG. 3.17 TYPICAL SOFTWARE DEFECT PROFILE
defects\textsuperscript{34}. There may be requirements defects, design defects, algorithmic processing defects, interface defects, performance defects, documentation defects, etc. Defects are also classified according to their severity, i.e., non-recoverable defects, serious defects (wrong answers), moderate defects (should be fixed for this release), and cosmetic defects (should be fixed for appearance reasons). The defects detected also tend to follow a Rayleigh/Norden distribution over the duration of the software life-cycle as shown in Fig.3.17. In turn, the product reliability is affected by the:

- size;
- application complexity;
- development productivity;
- project staff size; and
- development schedule.

As the product size increases, the absolute number of errors increase in an extremely non-linear way. More complex applications have lower productivity indices, require more time and effort, and have more defects. As the staff size increases, human communication paths increase leading to more ambiguities, confusion and defects. In order to improve the quality of the product one should attempt to:

- keep the product size small;
- invest in tools, methods, and management practices;
- use as small a team size as is possible;
- avoid delivering the product prematurely;
- measure and monitor defect rates;
- take action to correct processes in defect prone areas.

\textsuperscript{34} A defect is defined as a deviation from a specification.
3.10. System Dynamics

The principles and mechanics of System Dynamics were first worked out in the 1940's and 1950's, and ever since, the technique has been used to understand such diverse problems as technical obsolescence, urban decay, drug addiction, commodity price fluctuation, environmental deterioration and population growth.

Research findings over the past few years have shown that the decisions people make in organizations and the actions they choose to take are significantly influenced by the pressures, perceptions and incentives produced by the organizations' planning and control systems [WEI81]. The interpretations from such an interactive system, i.e., concrete results and conclusions, cannot be drawn by just our private analysis or mental models. The human mind is not adapted to correctly anticipate the dynamic consequences of interactions between the parts of complex societal systems [FOR71].

System Dynamics is the application of feedback control systems principles and techniques to management and organizational problems. It focusses on the structure\(^\text{35}\) and behavior of systems composed of interacting feedback loops. Component causal loop and flow diagrams offer a convenient way to represent loop structures before the development of system equations.

A system dynamics model is thus important because unlike a mental model, a system dynamics computer model can reliably trace through time, the implications of a messy maze of assumptions and interactions of a complex system [RIC81]. Large scale software projects are examples of such complex systems and system dynamics can

\(^{35}\) The system dynamics philosophy rests on a belief that the behavior of an organizational entity is principally caused by its inherent structure [ABD83].
thus play an important role in helping us study the software development process, its management, and its control.

The system dynamics approach begins with an effort to understand the system of forces that has created the problem and continues to sustain it. Relevant data are gathered from a variety of sources and as soon as a rudimentary measure of understanding is achieved, a formal model is developed, initially in the format of logical diagrams showing cause and effect relationships. The visual model is translated into a mathematically computational version. This model is then exposed to criticism, revised, and expressed again, and so on until a useful improved model is obtained. The process, in fact, generates a clearer understanding of the complex system or problem itself. The approach also forces the people involved to make explicit conjectures regarding the nature of the problem, the cause, alternative possible actions, and how various human, managerial, economic, and operational factors are interrelated. In short, the meticulous mathematical approach embedded in computer simulation turns out to be far superior than intuition about the probable consequences of proposed policies, and serves to supplement and correct human intuition [ROB81b]. System dynamics thus lends itself naturally to software project resource estimation, management and control, where there are numerous variable inputs to the system, which itself is complex and exhibits feedback behavior [ABD83].

In the past, system dynamics has been used to develop simulation models of the entire software development process [ABD90] [ABD84]. This chapter focusses on the development of a system dynamics simulation model for software project planning and control based on the Putnam modelling methodology discussed earlier in detail, while demonstrating the potential use of neural networks to provide
decision support to the management on estimates of "fuzzy" system variables.

3.10.0. Causal Loop Diagrams

Causal loops consist of causal relationships between variables which are defined explicitly in the system, and which interact with each other within the system. Causal loop relationships consist of positive and negative relationships. If a change in one variable generates a change in the same direction in another variable relative to its prior value, all other system variables remaining constant, the relationship between the two variables is positive. A negative relationship, characterized by a minus sign, occurs when a change in one variable produces a change in the opposite direction in the second variable, all other variables remaining unchanged. The assumption that all other impinging variables remain constant during the determination of causal polarity is central to causal analysis. Causal loop relationships when joined together form a feedback structure giving rise to causal loops.

To determine the polarity of a causal loop, we trace the consequences of an arbitrary change in one variable. For example in the causal loop of Fig.3.18, an increase in variable A causes an increase in variable B, which causes a decrease in variable C, which in turn cause variable A to decrease, thus giving a negative loop relationship: the loop attempts to maintain the variable at a specific goal despite external influences or perturbations to the contrary. When a feedback loop response to a variable change opposes the original perturbation, the loop is negative or goal seeking. When a loop response reinforces the original perturbation, the loop is positive.

(132)
FIG. 3.18 CAUSAL LOOP DIAGRAM

FIG. 3.19 GROWTH STRUCTURE FOR A TYPICAL VARIABLE (positive feedback)

FIG. 3.20 POSITIVE FEEDBACK FLOW DIAGRAM
3.10.1. Flow Diagrams

Causal loop diagrams lack the precision and detail of rate, level, and auxiliary elements found in flow diagrams. Flow diagrams bring out conceptualization errors that do not readily show up in causal loop diagrams. A causal loop diagram may inadvertently contain closed loops without levels, an error which would readily be pointed out by a flow diagram.

Flow diagrams also yield considerably more information than causal loop diagrams about the structure and behavior of systems. Because they depict the rate-level structure of a system, flow diagrams can also often indicate the different types of modes of systems behavior.

3.10.2. Positive Feedback Structure

In a positive feedback process, a variable continually feeds back on itself to reinforce its own growth or collapse, and is often referred to as the "bandwagon" or "snowball" effect. Both "vicious circles" and "virtuous circles" are synonyms for positive feedback. In a vicious circle, the worsening of one element in the causal chain brings about further degradation of that element. Conversely, in a virtuous circle, positive changes in a system element trigger further improvement.

Most positive feedback systems are characterized by an exponential growth curve as shown in Fig. 3.19. World population, industrial growth, and food production, all exhibit this kind of growth.

The general structure of a simple positive feedback system is shown in Fig. 3.20. A one way flow of material accumulates in the level
LEV. In turn, information about the quantity in the level at any time controls the flow into LEV, via the rate RT. RT is related to LEV via a proportionality constant CONST. The DYNAMO equations for the system (arbitrary CONST = 0.2), are:

\[
\begin{align*}
\text{LEV}.K &= \text{LEV}.J + (\text{DT})(\text{RT}.JK) \quad \text{(3.29)} \\
\text{LEV} &= 1.0 \quad \text{(3.30)} \\
\text{RT}.KL &= \text{CONST} \times \text{LEV}.K \quad \text{(3.31)} \\
\text{CONST} &= 0.2 \quad \text{(3.32)} \\
\text{DT} &= 1.0 \quad \text{(3.33)}
\end{align*}
\]

Numerical simulation of these equations over multiple iterations in time indicate exponential growth inherent in the system. An analytical solution to the system is of the form:

\[
\text{LEV}(t) = \text{LEV}(0) \exp(\text{CONST} \times t) \quad \text{(3.34)}
\]

where \(\text{LEV}(0)\) and \(\text{LEV}(t)\) are values of the level at the initial simulation timepoint and any time \(t\) respectively. The time constant of the system is defined as the reciprocal of CONST or \(T = 1/\text{CONST}\). The time constant has the dimension of the units of time. The doubling time is a period of time over which the level doubles its value, and is \(0.69T\) or approximately 70% of the time constant. The equations for positive feedback structure show that any level value initially greater than zero produces a positive rate value which in the next DT period produces a still larger value.

3.10.3. Negative Feedback Structure

Negative feedback is characterized by goal-directed or goal-oriented behavior. Any controlled system or implications of control itself entails goal orientation. Terms such as self-governing, self-regu-
ating, self-equilibrating, homeostatic, or adaptive, all implying the presence of a goal, describe negative feedback. The causal loop diagram of a simple negative feedback system is shown in Fig.3.21. The four basic elements of negative feedback are: the desired state (goal); the discrepancy; the action; and the system state (level). The difference between a negative feedback loop and a positive feedback loop lies entirely in the decision process sector: information sensing, comparison, and decision making components which intervene between the system state and action. The simplest negative feedback system contains a goal and discrepancy element in the decision process sector. The goal serves as the reference or guideline on which the system bases its action.

The generalized flow diagram of Fig.3.22 shows a negative feedback structure. Assuming a goal GL, the rate RT provides the action component of the system. The discrepancy DISC, an auxiliary variable, and the fraction of the discrepancy acted upon per time unit, FPT, a constant, control the rate RT. The DYNAMO equations for this system are:

\[
LEV.K = LEV.J + (DT) (RT.JK) \\
LEV = 0 \\
RT.KL = FPT \times DISC.K \\
FPT = 0.1 \\
DISC.K = GL - LEV.K \\
GL = 100 \\
\]

\[(3.35)\] \[(3.36)\] \[(3.37)\] \[(3.38)\] \[(3.39)\] \[(3.40)\]

The behavior of the system with respect to time can be seen by simulation as was done in the case of positive feedback.
**Fig. 3.21**: Causal Loop Diagram of Negative Feedback Structures

**Fig. 3.22**: Generalized Flow Diagram for Negative Feedback System.
The analytical expression representing system behavior, as a function of time and constants FPT and GL is:

\[ \text{LEV}(t) = GL + [\text{LEV}(0) - GL] \exp(-FPT \cdot t) \quad \ldots (3.41) \]

The time constant of the system is given by $1/FPT$. In a period of time roughly equal to five time constants, the system attains 99.99% of its goal.

Other negative feedback loop structures are the zero value goal structure and the system compensation structure in the presence of a constant inflow or outflow rate over which the system exerts no control as shown in Fig. 3.23 and Fig. 3.24 respectively. The system equations for the system compensation structure can be written in the same manner as before with the only difference that the level LEV will now be controlled by two rates and the system accumulates a net rate (inflow or outflow, depending on the dominant rate), and acquires a new goal value (larger or smaller, depending on the rates again). The negative feedback system compensates for the additional inflow or outflow rate by attaining an equilibrium value different from the desired one.

The negative feedback system behavior thus has two separable regions, the transient region and the steady state region, and displays asymptotic behavior in nature. The level is characteristically goal seeking and transitory.

3.10.4. S-Shaped Growth Structures

The S-shaped growth structure, also known as "logistic" or sigmoidal growth, is a combination of both exponential and asymptotic growth. The time path includes two distinct regions: exponential growth,
FIG. 3.23 ZERO GOAL STRUCTURE FLOW DIAGRAM

FIG. 3.24 INFLOW-OUTFLOW STRUCTURE FLOW DIAGRAM
the result of positive feedback; followed by asymptotic growth, the result of negative feedback; as shown in Fig. 3.25. A sigmoidal trend characterizes physical and mental development of an individual, learning curves, diffusion phenomenon in a fixed population, etc.

The S-shaped growth structure system flow diagram is shown in Fig. 3.26. There has to be a shift in loop predominance from positive to negative. The rate-level relationship has two distinct curves, one with a positive slope, typical of positive feedback; and a curve with negative slope, characteristic of negative feedback. The DYNAMO equations for the system are as follows (RTV is the rate value, indexed from RTT, a lookup rate table):

\[
\begin{align*}
\text{LEV}.K &= \text{LEV}.J + (\text{DT}) (\text{RT}.JK) \\
\text{LEV} &= 1 \\
\text{RT}.KL &= \text{RTV}.K \\
\text{RTV}.K &= \text{TABLE}(\text{RTT}, \text{LEV}.K, 0, 1200, 100) \\
\text{RTT} &= 0/5/10/15/20/25/20/15/10/5/0/-5/-10
\end{align*}
\]

The behavior of the level over time can be understood by examining the rate level table. Increasing rate RT values generate increasing level LEV values till a maximum value of the rate is reached, where constraints begin to appear that suppress the growth rate. The next increment to the level moves the level LEV into the negative feedback region of the rate level so that the level continues to increase, but at a decreasing rate. As the rate now approaches zero, the system approaches its goal.

Prior to growth when the level value is zero, the system is in "unstable equilibrium". Unless exogenously disturbed, the system remains in unstable equilibrium. A disturbance such as a minimal
FIG. 3.25 S-SHAPED GROWTH STRUCTURE

FIG. 3.26 S-SHAPED STRUCTURE FLOW DIAGRAM
increase in LEV produces an RT value other than zero, and the system level increases with S-shaped growth behavior.

Thus, a simple rate-level structure that first exhibits rate increase with rising level values, and then rate decrease with further level growth, produces the necessary shift in dominance from positive to negative feedback, which typically characterizes sigmoidal feedback growth.

3.11. Modelling the Putnam Approach with System Dynamics

System dynamics can prove to be an extremely powerful tool if one wants to gain insight into the dynamic aspects of a system. As has already been discussed, the software cost estimation and control process is extremely dynamic and time-sensitive. There are a number of interacting factors that cause the process to have such a nature which have been highlighted during the course of the earlier discussion. Before proceeding to develop a comprehensive simulation model, it is necessary to understand the causal relationships that exist between different variables of the software system. These component variables have been identified as follows:

- total Generic life-cycle effort, $K$
- Development sub-cycle effort, $K_d$
- minimum development time, $t_d^{(\text{min})}$
- development time or the Generic life-cycle peak time, $t_d$
- Development sub-cycle peak time, $t_{\text{pp}}$
- size of the system, $SS$
- application functionality $F$
- manpower buildup, $D_0$
- Generic life-cycle peak manning, $m_0$

(142)
Development sub-cycle peak manning, $m_{0d}$
organizational productivity parameter, Pr
code production rate, CPR
defect rate, DR
reliability of the system, REL

3.11.0. Model Causal Loop Diagram

The fundamental causal relationships between pairs of variables can be found by assuming only one of the variables to change, all others remaining static, and these relationships can be summarized as follows:

- if the functionality, $F$, increases, then the number of source statements, $SS$, also increase (Fig.3.27(a));

- if the number of source statements, $SS$, increase, the Generic life-cycle effort, $K$, increases (Fig.3.27(b));

- if the number of source statements, $SS$, increase, the defect rate, DR, increases (Fig.3.27(c));

- if the Generic life-cycle effort, $K$, increases, the minimum development time, $t_d^{(\text{min})}$, decreases, and if $t_d^{(\text{min})}$ increases, $K$ decreases (Fig.3.27.(d));

- if the Generic life-cycle effort, $K$, increases, the Development sub-cycle effort, $K_d$, also increases (Fig.3.27(e));
FIG. 3.27 COMPONENT VARIABLE CAUSAL RELATIONSHIPS.
- if the Generic life-cycle peak time, \( t_d \), increases, the Development sub-cycle peak time, \( t_{0d} \), also increases (Fig.3.27(f));

- if the minimum development time, \( t_d^{(\text{min})} \), increases, the defect rate, DR, decreases (Fig.3.27(g));

- if the organizational productivity, \( Pr \), increases, the code production rate, CPR, also increases (Fig.3.27(h));

- if the code production rate, CPR, increases, the Generic life-cycle effort, \( K \), decreases (Fig.3.27(i));

- if the functionality, \( F \), increases, the manpower buildup, \( D_0 \), also increases, (Fig.3.27(j));

- if the manpower buildup, \( D_0 \), increases, the Generic life-cycle peak manning, \( m_0 \), also increases (Fig.3.27(k));

- if the Generic life-cycle peak manning, \( m_0 \), increases, the Development sub-cycle peak manning, \( m_{0d} \), also increases (Fig.3.27(l));

- if the Generic life-cycle peak manning, \( m_0 \), increases, the Generic life-cycle effort, \( K \), also increases (Fig.3.27(m));

- if the Development sub-cycle peak manning, \( m_{0d} \), increases, the Development sub-cycle effort, \( K_d \), also increases (Fig.3.27(n));
if the Generic life-cycle peak manning, $m_g$, increases, the Generic life-cycle peak time, $t_d$, decreases (Fig. 3.27(o));

if the Development sub-cycle peak manning, $m_{0d}$, increases, the Development sub-cycle peak time, $t_{0d}$, decreases (Fig. 3.27(p));

if the Generic life-cycle peak time, $t_d$, increases, the defect rate, $DR$, is expected to decrease (Fig. 3.27(q));

if the Development sub-cycle peak time, $t_{0d}$, increases, the defect rate, $DR$, is expected to decrease (Fig. 3.27(r));

if the defect rate, $DR$, increases, the reliability of the system, $REL$, decreases (Fig. 3.27(s));

if the Development sub-cycle effort, $K_d$, increases, the defect rate, $DR$, increases (Fig. 3.27(t));

if the productivity $Pr$, increases, the defect rate, $DR$ can be expected to decrease (Fig. 3.27(u)).

These causal relationships between different pairs of variables can be integrated into one comprehensive causal loop diagram as shown in Fig. 3.28. The loop interactions are evidently clear from the diagram, and inter-variable dependencies are highlighted.

3.11.1. Static Model Flow Diagram

The flow diagram for the Putnam estimation technique can be developed on the basis of the understanding of the process itself as...
FIG. 3.28. COMPREHENSIVE CAUSAL LOOP DIAGRAM
depicted in Fig. 3.11, and the variable causal relationships as identified in Fig. 3.27 and 3.28.

The entire flow-diagram can be divided into two fundamental kinds of estimation simulation models: static and dynamic. The static simulation model has three fundamental sub-models: (i) the productivity estimation sub-model; (ii) the software sizing sub-model; and (iii) the management enterprise sub-model.

The productivity estimation sub-model as shown in Fig. 3.29, helps simulate the productivity estimation process in an enterprise. As depicted in the model, there are baseline statistics available from which the development effort, $K_d$, development time, $t_d$, system size, $SS$, and size related constant, $b$, can be found to estimate the productivity of an organization on a previously completed project. This productivity estimate is used to find the average productivity parameter of the organization on that class of projects. DYNAMO equations can be developed as follows, and the simulation is conducted over project indices, rather than time (which is why it falls into the static sub-model category):

\[
\begin{align*}
\text{TOTALPR.H} &= \text{TOTALPR.G} + (\text{DINDEX})(\text{RT.GH}) \quad \ldots \quad (3.47) \\
\text{TOTALPR.O} &= 0 \quad \ldots \quad (3.48) \\
\text{DINDEX} &= 1 \quad \ldots \quad (3.49) \\
\text{RT.UV} &= \text{PR.U} \quad \ldots \quad (3.50) \\
\text{PR.U} &= \frac{\text{SIZE.U}}{\left\{ \left[ K.U \right]^{1/3} \* \left[ T_d.U \right]^{4/3} \right\}} \quad \ldots \quad (3.51) \\
\text{PRPARAM.U} &= \frac{\text{TOTALPR.U}}{U} \quad \ldots \quad (3.52) \\
\text{PINDEX} &= \text{DATA.U} \quad \ldots \quad (3.53) \\
\text{DATA.U} &= \text{TABLE(DATAT, PRPARAM.U, 754, 987, 1220, 1597, ..., 28657, 35422, 46368, 57314, 75025, 92736, 121393)} \quad \ldots \quad (3.54) \\
\text{DATAT} &= 1/2/3/4/.../16/17/18/19/20/21/22 \quad \ldots \quad (3.55)
\end{align*}
\]
FIG. 3.29 PRODUCTIVITY ESTIMATION FLOW MODEL

FIG. 3.30 SYSTEM SIZE ESTIMATION FLOW MODEL
The second sub-model shown in Fig. 3.30, relates to the software size estimation technique based on a Delphi polling approach. In this, depending upon the application complexity and expert opinion, preliminary module partitioning is performed, and after deciding reusable code issues for each of the functional modules, the lowest, most likely, and highest estimates for the code along with their uncertainties are used to estimate the module code size and the associated standard deviation. This is then added into the total size of the project and the total standard deviation. The associated DYNAMO equations are:

\[
\begin{align*}
SS.K &= SS.J + (\text{DINDEX})(RTSS.JK) \quad \ldots (3.56) \\
\text{SIGMASQR.K} &= \text{SIGMASQR.J} + (\text{DINDEX})(\text{RTSIGMA.JK}) \quad \ldots (3.57) \\
\text{DINDEX} &= 1 \quad \ldots (3.58) \\
\text{RTSS.JK} &= \text{SSMODULE.K} \quad \ldots (3.59) \\
\text{RTSIGMA.JK} &= (\text{SIGMAMODULE.K})^2 \quad \ldots (3.60) \\
\text{SSMODULE.K} &= \frac{\mid SL.K + 4 SM.K + SH.K \mid}{6} \quad \ldots (3.61) \\
\text{SIGMAMODULE.K} &= \frac{SH.K - SL.K}{6} \quad \ldots (3.62) \\
\text{SIGMASYSTEM} &= (\text{SIGMASQR.K})^{1/2} \quad \ldots (3.63) \\
\text{SLOCSYSTEM} &= SS.K \quad \ldots (3.64)
\end{align*}
\]

3.11.2. Using Neural Networks to Aid Management Decisions

The third static sub-model is concerned with the construction of a planning zone for the project after taking into account the various management constraints. At the center of this sub-model lies the management, which would comprise essentially those who are responsible for the management and control of resources within the development organization, policy matters, and high level user interaction. Fig. 3.31 conveys a typical view of the different
FIG. 3.31 MANAGEMENT INTERACTIONS FLOW MODEL
management interactions that take place in the process of deciding upon a suitable planning point \( P \).

As far as the scope of the cost estimation model is concerned, the user interacts with the management primarily to decide the application, provide the system requirements, and the earliest and latest delivery dates, \( t_{\text{min}} \) and \( t_{\text{max}} \) respectively. The management then takes its decisions on critical project constraints. Such constraints necessitate the setting up of a specific manpower loading profile (manpower-buildup index (MBI)). This involves establishing a cost ceiling and deciding on the overall feasibility of the project in terms of the available manpower, acceptable schedules, project difficulty, technical knowhow for the application, monetary resource availability and various human factors.

Thus, the major constraints to be considered in the decision making process are:

(i) user delivery date  
(ii) cost ceiling  
(iii) maximum manpower available  
(iv) risk considerations (and thereby resultant average defect rate).

These factors, together with the organizational application productivity and the system size estimate, are then used to construct a planning zone from which a suitably safe planning point, \( P \), is selected. The planning point decides the Generic life-cycle effort, \( K \), the Development sub-cycle effort, \( K_d \), and the development time \( t_d \). These data along with the size and productivity estimates are then used for dynamic simulation of the model.

(152)
It has been discussed earlier that the productivity index and the SLOC are estimated using the historical data of the firm and expert Delphi polling, and are direct inputs to the software equation (eqn. 3.23). The second fundamental equation is the MBI equation (eqn. 3.7) which helps select a suitable manpower buildup over the duration of the project under consideration. When these two equations are solved simultaneously, they yield a minimum time solution which decides the peak manning $m_0$ which derives from the Rayleigh-Norden distribution of manpower (eqn.3.5).

The manpower buildup parameter is governed primarily by three factors:

(i) the task concurrency of the project (a high task concurrency implies a high manpower buildup index);

(ii) the application complexity (real-time systems with many interactions (low MBI) to business applications built from reusable code modules (high MBI));

(iii) schedule pressures of the organization (a shortage of time for a particular project for which the delivery dates are not extendable would have a high MBI).

One can easily see that these three factors are difficult to quantify objectively, and can be better described using "fuzzy" ranges of values such as "low", "medium" or "high". Estimation of the manpower buildup index thus involves a somewhat intuitive decision-making process, something that is readily supported by neural networks. The manpower buildup has strong implications for the system delivery date, project cost, manpower requirements and risk. It is therefore necessary that the estimation of this parameter be done correctly.
3.11.3. Neural Network Decision Support Model

An example model for decision support using neural networks has been illustrated in Fig. 3.32. Various interacting factors are linked together with the help of a neural network ANN and a decision support system, DSS. ANN is a feedforward neural network trained using backpropagation [RUM86] [MCL88], and DSS is a decision support system capable of taking logical decisions.

The feedforward neural network, ANN, accepts "intuitively" judged inputs regarding the task concurrency, schedule pressure and the inverse application complexity, to give the best estimate for the MBI that should be selected based on its learned knowledge. For this purpose, the inputs have been divided into four categories - low (L), low-medium (LM), medium-high (MH), and high (H). Table III.6 shows how such a categorization can be effected for each of the three inputs. For each possible combination a specific desired output manpower buildup index is chosen, yielding 64 possible cubic decision regions, examples of which are shown in Fig. 3.33. The desired MBI for each of the cubes is decided by letting the cube closest to the origin have an MBI of 1, and the cube farthest from the origin have an MBI of 6. Elsewhere in the region, the MBI's are distributed as shown in Table III.7.

In order to train the system effectively, it is necessary to first make a suitable selection regarding the number of nodes that should be selected in the hidden layer of the neural network. In accordance
FIG. 3.32 MANAGEMENT DECISION SUPPORT MODEL WITH ANN AND DSS
Table III.6 "Fuzzy" partitioning of inputs to the feedforward neural network

<table>
<thead>
<tr>
<th>Inverse Application Complexity (IAC)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L :</td>
<td>Entirely new software with many interfaces and interactions with low MBI implications</td>
</tr>
<tr>
<td>LM :</td>
<td>New stand-alone systems with high functionality with low-medium MBI implications</td>
</tr>
<tr>
<td>MH :</td>
<td>New stand-alone systems with low functionality with medium-high MBI implications</td>
</tr>
<tr>
<td>H :</td>
<td>Software re-built from re-usable code with high MBI implications</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task Concurrency (TC)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L :</td>
<td>Low task concurrency</td>
</tr>
<tr>
<td>LM :</td>
<td>Low to medium task concurrency</td>
</tr>
<tr>
<td>MH :</td>
<td>Medium to high task concurrency</td>
</tr>
<tr>
<td>H :</td>
<td>High task concurrency</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Schedule Pressure (SP)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>L :</td>
<td>Low schedule pressure</td>
</tr>
<tr>
<td>LM :</td>
<td>Low to medium schedule pressure</td>
</tr>
<tr>
<td>MH :</td>
<td>Medium to high schedule pressure</td>
</tr>
<tr>
<td>H :</td>
<td>High schedule pressure</td>
</tr>
</tbody>
</table>

with Mirchandani's theorem\(^{36}\), if \( h = 6 \) then \( M(h,d) = 42 \), which is less than the desired 64 partitions. A network with 6 hidden nodes will therefore not be able to create the desired number of partitions. If \( h \) is chosen as 7, \( M(h,d) = 64 \). This partition matches that which is desired and therefore suffices in the present case. Fig.3.34 shows the structure of the network selected in accordance

\[^{36}\text{For this purpose we may refer to a theorem by Mirchandani and Cao [MIR89] which states that the maximum number of decision regions, } M, \text{ that can be created for a d dimensional input, with h hidden nodes in a single hidden layer feedforward neural network is given by :}\]

\[
M(h,d) = \sum_{k=0}^{d} \binom{h}{k} \cdot \frac{h!}{(h-k)! k!} \cdot \left( \frac{h}{k} \right) - 0, h < k
\]

(156)
<table>
<thead>
<tr>
<th>IAC</th>
<th>TC</th>
<th>SP</th>
<th>MBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>1</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>LM</td>
<td>2</td>
</tr>
<tr>
<td>LM</td>
<td>L</td>
<td>L</td>
<td>2</td>
</tr>
<tr>
<td>L</td>
<td>MH</td>
<td>L</td>
<td>2</td>
</tr>
<tr>
<td>L</td>
<td>LM</td>
<td>LM</td>
<td>2</td>
</tr>
<tr>
<td>LM</td>
<td>L</td>
<td>LM</td>
<td>2</td>
</tr>
<tr>
<td>LM</td>
<td>LM</td>
<td>LM</td>
<td>3</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>3</td>
</tr>
<tr>
<td>L</td>
<td>LM</td>
<td>MH</td>
<td>3</td>
</tr>
<tr>
<td>LM</td>
<td>L</td>
<td>MH</td>
<td>3</td>
</tr>
<tr>
<td>LM</td>
<td>LM</td>
<td>LM</td>
<td>3</td>
</tr>
<tr>
<td>L</td>
<td>MH</td>
<td>MH</td>
<td>4</td>
</tr>
<tr>
<td>LM</td>
<td>MH</td>
<td>L</td>
<td>4</td>
</tr>
<tr>
<td>L</td>
<td>LM</td>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>LM</td>
<td>L</td>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>LM</td>
<td>4</td>
</tr>
<tr>
<td>LM</td>
<td>LM</td>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td>LM</td>
<td>LM</td>
<td>4</td>
</tr>
<tr>
<td>LM</td>
<td>MH</td>
<td>MH</td>
<td>4</td>
</tr>
<tr>
<td>L</td>
<td>MH</td>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>MH</td>
<td>L</td>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>MH</td>
<td>4</td>
</tr>
<tr>
<td>MH</td>
<td>MH</td>
<td>MH</td>
<td>5</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>LM</td>
<td>MH</td>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>MH</td>
<td>LM</td>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>H</td>
<td>LM</td>
<td>MH</td>
<td>5</td>
</tr>
<tr>
<td>MH</td>
<td>MH</td>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>H</td>
<td>MH</td>
<td>MH</td>
<td>5</td>
</tr>
<tr>
<td>H</td>
<td>LM</td>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>MH</td>
<td>6</td>
</tr>
<tr>
<td>MH</td>
<td>H</td>
<td>H</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IAC</th>
<th>TC</th>
<th>SP</th>
<th>MBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>LM</td>
<td>L</td>
<td>2</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>MH</td>
<td>2</td>
</tr>
<tr>
<td>MH</td>
<td>L</td>
<td>L</td>
<td>2</td>
</tr>
<tr>
<td>LM</td>
<td>LM</td>
<td>L</td>
<td>2</td>
</tr>
<tr>
<td>L</td>
<td>MH</td>
<td>MH</td>
<td>3</td>
</tr>
<tr>
<td>LM</td>
<td>MH</td>
<td>L</td>
<td>3</td>
</tr>
<tr>
<td>LM</td>
<td>LM</td>
<td>LM</td>
<td>3</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>MH</td>
<td>3</td>
</tr>
<tr>
<td>LM</td>
<td>MH</td>
<td>L</td>
<td>3</td>
</tr>
<tr>
<td>LM</td>
<td>LM</td>
<td>LM</td>
<td>3</td>
</tr>
<tr>
<td>L</td>
<td>MH</td>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>LM</td>
<td>L</td>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>LM</td>
<td>4</td>
</tr>
<tr>
<td>LM</td>
<td>LM</td>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td>LM</td>
<td>LM</td>
<td>4</td>
</tr>
<tr>
<td>LM</td>
<td>MH</td>
<td>MH</td>
<td>4</td>
</tr>
<tr>
<td>L</td>
<td>MH</td>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>MH</td>
<td>L</td>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>MH</td>
<td>4</td>
</tr>
<tr>
<td>MH</td>
<td>MH</td>
<td>MH</td>
<td>4</td>
</tr>
<tr>
<td>L</td>
<td>MH</td>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>MH</td>
<td>L</td>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>MH</td>
<td>MH</td>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>H</td>
<td>MH</td>
<td>HM</td>
<td>5</td>
</tr>
<tr>
<td>H</td>
<td>LM</td>
<td>H</td>
<td>5</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>MH</td>
<td>6</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
<td>6</td>
</tr>
</tbody>
</table>

\* Inverse Application Complexity
\* Schedule pressure
\* Task concurrency
\* Manpower Buildup Index
FIG. 3.33 MBI DECISION SPACE PARTITIONING

FIG. 3.34 STRUCTURE OF FEEDFORWARD NEURAL NETWORK ANN FOR MBI ESTIMATION
with this theorem. A minimum of 64 training patterns, \{T\}, will be required to train this system. Network training statistics have been discussed in Appendix IV.

DSS is a logical decision support system as shown in Fig. 3.35. Depending upon the status of the peak manpower \(m_o\), the cost of the project \(C\), the development time \(t_d\), and the risk factor \(r\), the management may decide to change one or more of (i) the task concurrency (by changing the number of tasks being done simultaneously), (ii) the application complexity (by delivering a system on time with fewer functions completed), and (iii) the schedule pressure (by asking the user to extend the delivery date). All three factors, if changed appropriately, will help reduce the MBI.

In Fig. 3.35, \(D_1\), \(D_2\), \(D_3\) and \(D_4\), are ceiling violation indicators, and respond if any of (i) the peak manning, (ii) the cost, (iii) the development time, or (iv) the risk exceed their acceptable limits. \(D_1\) responds if the peak manning exceeds its maximum allowed value, \(m_{\text{max}}\); \(D_2\) responds if the cost exceeds the maximum available resources, \(C_{\text{max}}\); \(D_3\) responds if the development time exceeds its maximum allowed value, \(t_{\text{max}}\); and \(D_4\) responds if the risk factor exceeds the maximum allowable limit, \(r_{\text{max}}\).

---

37 To generate training patterns from Table III.7, the center point of each micro-cube partition was considered as the training pattern for that specific micro-cube. Assuming that the macro-cube scale varies from 0 to 1, the four partitions along each of the three axes become 0-0.25, 0.26-0.5, 0.51-0.75, and 0.76-1.00. For example, for the cube closest to the origin the desired MBI is 1, and the three inputs for the training pattern become IAC = 0.125, TC = 0.125, SP = 0.125.

38 In this context, the risk factor, for a particular development time \(t_d\), is the probability that the project would exceed the delivery date, \(t_{\text{max}}\), as demanded by the customer. This is calculated by conducting a Monte Carlo analysis of \(X\) and \(t_d\) with all possible uncertainties in the inputs included. However, the risk can also be thought of in terms of the ratio of the minimum development time, \(t_d\), to the maximum delivery date, \(t_{\text{max}}\). As \(t_d\) approaches \(t_{\text{max}}\) the risk increases.
FIG. 3.35 DECISION SUPPORT SYSTEM DSS ARCHITECTURE
The details of the decision logic have been discussed in Table III.8 ahead. For example, consider the following case:

If \([C > C_{\text{MAX}}] \text{ and } [t_d > t_{\text{MAX}}] \text{ and } [m_0 < m_{\text{MAX}}] \text{ and } [r < r_{\text{MAX}}]\)

This is a conflict situation: decreasing \(t_d\) increases \(C\), and vice-versa. To resolve this conflict, one can either ask the user to extend the latest delivery date, \(t_{\text{MAX}}\), or reduce the targeted functionality, so that the above conditions are no longer violated. In the latter case, reducing the functionality would allow for reduced SLOC which would in turn imply the same \(t_d\) at a reduced MBI, i.e., a lower projected cost. However, increasing the delivery date would prevent the \(t_{\text{MAX}}\) condition from being violated, while at the same time allowing for a reduced defect rate. The cost may also then be reduced by either reducing the task concurrency or the application complexity, or both. In either case, the peak Manning gets reduced.

In the case of a persisting conflict situation, one may have to attempt to resolve the issue through long term changes such as organizational productivity increases (possibly through the acquisition of newer technologies), or modifications in the overall managerial structure of the organization.

The management model incorporating both the neural network as well as the decision support system has been depicted in detail in Fig.3.32.

3.11.4. The Dynamic Model Flow Diagram

The dynamic sub-models relate to three different but related aspects of the cost estimation process. Simulation in these models is
### TABLE III.8 Recommended Action on Violation of Management Number Ceilings

<table>
<thead>
<tr>
<th>#</th>
<th>C</th>
<th>( t_d )</th>
<th>( m_0 )</th>
<th>r</th>
<th>MBI</th>
<th>TC</th>
<th>F*</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>NE</td>
<td>NE</td>
<td>NE</td>
<td>NE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>No action</td>
</tr>
<tr>
<td>2.</td>
<td>NE</td>
<td>NE</td>
<td>NE</td>
<td>E</td>
<td>INC</td>
<td>INC</td>
<td></td>
<td>Increasing the TC reduces ( t_d ), which reduces the risk.</td>
</tr>
<tr>
<td>3.</td>
<td>NE</td>
<td>NE</td>
<td>E</td>
<td>NE</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>Decreasing TC or F or both will ease the peak manning.</td>
</tr>
<tr>
<td>4.</td>
<td>NE</td>
<td>NE</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
<td>DEC</td>
<td>Conflict situation: decreasing ( m_0 ) increases r and vice-versa.</td>
</tr>
<tr>
<td>5.</td>
<td>NE</td>
<td>E</td>
<td>NE</td>
<td>NE</td>
<td>INC</td>
<td>INC</td>
<td>DEC</td>
<td>Increasing TC and/or decreasing F would help reduce ( t_d ).</td>
</tr>
<tr>
<td>6.</td>
<td>NE</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>INC</td>
<td>INC</td>
<td>DEC</td>
<td>As in (5): reducing ( t_d ) also reduces the risk.</td>
</tr>
<tr>
<td>7.</td>
<td>NE</td>
<td>E</td>
<td>E</td>
<td>NE</td>
<td></td>
<td></td>
<td>DEC</td>
<td>Conflict situation: decreasing ( t_d ) increases ( m_0 ) and vice-versa.</td>
</tr>
<tr>
<td>8.</td>
<td>NE</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
<td>DEC</td>
<td>Conflict situation: as in (4)/(8) above.</td>
</tr>
<tr>
<td>9.</td>
<td>E</td>
<td>NE</td>
<td>NE</td>
<td>NE</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>Decreasing TC and/or F reduces ( m_0 ) which reduces the overall cost.</td>
</tr>
<tr>
<td>10.</td>
<td>E</td>
<td>NE</td>
<td>NE</td>
<td>E</td>
<td></td>
<td></td>
<td>DEC</td>
<td>Conflict situation: if the cost is reduced by reducing TC, ( t_d ) would increase, thereby increasing the risk.</td>
</tr>
<tr>
<td>11.</td>
<td>E</td>
<td>NE</td>
<td>E</td>
<td>NE</td>
<td>DEC</td>
<td>DEC</td>
<td></td>
<td>Decreasing TC reduces both the cost and the peak manning.</td>
</tr>
<tr>
<td>12.</td>
<td>E</td>
<td>NE</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
<td>DEC</td>
<td>Conflict situation as discussed in (4)/(8)/(12) above.</td>
</tr>
<tr>
<td>13.</td>
<td>E</td>
<td>E</td>
<td>NE</td>
<td>NE</td>
<td></td>
<td></td>
<td>DEC</td>
<td>Conflict situation: if the cost is reduced, ( t_d ) increases, and vice-versa.</td>
</tr>
<tr>
<td>14.</td>
<td>E</td>
<td>E</td>
<td>NE</td>
<td>E</td>
<td></td>
<td></td>
<td>DEC</td>
<td>Conflict situation, as discussed in (13) above.</td>
</tr>
<tr>
<td>15.</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>NE</td>
<td></td>
<td></td>
<td>DEC</td>
<td>Conflict situation as discussed in (13) above.</td>
</tr>
<tr>
<td>16.</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td></td>
<td></td>
<td>DEC</td>
<td>Conflict situation as in (12)-(15) above.</td>
</tr>
</tbody>
</table>

**E**: Exceeds  
**NE**: Does not exceed  
**INCREASE**: Increase  
**DEC**: Decrease  
**F**: Functionality

(162)
conducted over the life-cycle period of the software system by choosing a suitable increment DT in the simulation procedure.

The first dynamic sub-model relates to total effort (cost) simulation using the manning rate as shown in Fig. 3.36. The total effort (cost) C(t) is the LEVEL variable, and the manning m(t) is the rate variable. The simulation can also be conducted for the development sub-cycle by using the appropriate parameters C_d(t) and m_d(t), which has also been indicated in the same figure.

The DYNAMO equations for this simulation are:

\[ \text{TOTALEFFORT}.H = \text{TOTALEFFORT}.G + \text{DT} \left( \text{RT.GH} \right) \quad \ldots (3.65) \]
\[ \text{DT} = 0.1 \text{ YR} \quad \ldots (3.66) \]
\[ \text{RT.UV} = \text{MANNING}.U \quad \ldots (3.67) \]
\[ \text{MANNING}.U = \]
\[ \left( \frac{\text{CUMEFFORT}}{\text{T}_{d}^2} \right) .U \ast \text{U(DT)} \ast \exp \left( \frac{-\left( \text{U*DT} \right)^2}{\left( 2\text{T}_{d}^2 \right)} \right) \quad \ldots (3.68) \]
\[ \text{CUMEFFORT} = K \quad \ldots (3.69) \]

and

\[ \text{TOTALDEVEFFORT}.H = \text{TOTALDEVEFFORT}.G + \text{DT} \left( \text{RT.GH} \right) \quad \ldots (3.70) \]
\[ \text{DT} = 0.1 \text{ YR} \quad \ldots (3.71) \]
\[ \text{RT.UV} = \text{DEVMANNING}.U \quad \ldots (3.72) \]
\[ \text{DEVMANNING}.U = \]
\[ \left( \frac{\text{CUMDEVEFFORT}}{\text{T}_{0d}^2} \right) .U \ast \text{U(DT)} \ast \exp \left( \frac{-\left( \text{U*DT} \right)^2}{\left( 2\text{T}_{0d}^2 \right)} \right) \quad \ldots (3.73) \]
\[ \text{CUMDEVEFFORT} = K_d \quad \ldots (3.74) \]

Cumulative cost figures and manning profile data are sent back to the management for consideration and monitoring of progress on the project. The simulation time ST is also sent back to the management so that milestone slippage can be noticed.
FIG. 3.36 CUMULATIVE COST SIMULATION FLOW MODEL
The second sub-model is shown in Fig.3.37. This model simulates the code production over the development life-cycle. The level variable is the code produced, and the rate variable is the code production rate, CPR. The development life-cycle effort, $K_d$, the productivity, $Pr$, and the development sub-cycle peak time, $t_{Od}$, are used to calculate the CPR.

The DYNAMO equations for this sub-model are:

\[
\begin{align*}
\text{SLOC}.H &= \text{SLOC}.G + DT \times (\text{RT}.GH) \quad \cdots (3.75) \\
\text{DT} &= 0.1 \times \text{YR} \quad \cdots (3.76) \\
\text{RT}.UV &= \text{CPR}.U \quad \cdots (3.77) \\
\text{CPR}.U &= Pr \times \left(\frac{K_d}{T_{Od}}\right)^2 \times U(DT) \times \exp\left(-\frac{(U \times DT)^2}{2T_{Od}^2}\right) \quad \cdots (3.78)
\end{align*}
\]

The SLOC produced at any instant of simulation time is fed back to the management to decide the progress on the code development of the project.

The third sub-model shown in Fig.3.38 concerns the defects or errors that are discovered due to incorrect coding of modules. The level variable is the total number of errors discovered, and the rate variable is the defect rate, DR. The data on the planning point and the size of the software system is used to generate a defect rate equation similar to the Rayleigh/Norden profile, and the defect occurrences are simulated.

The DYNAMO equations for this simulation are:

\[
\begin{align*}
\text{DEFECTS}.H &= \text{DEFECTS}.G + DT \times (\text{RT}.GH) \quad \cdots (3.79) \\
\text{DT} &= 0.1 \times \text{YR} \quad \cdots (3.80) \\
\text{RT}.UV &= \text{DR}.U \quad \cdots (3.81) \\
\text{DR}.U &= \text{RN}.U \quad \cdots (3.82)
\end{align*}
\]

(165)
FIG. 3.37 CODE PRODUCTION FLOW MODEL

FIG. 3.38 DEFECTS FLOW MODEL
The number of defects that are expected to occur is then used to calculate the reliability of the software system. This reliability is calculated as the mean-time-to-failure (MTTF), and is inversely proportional to the number of defect occurrences. The reliability data is then fed back to the management, which can then take recourse to measures to improve the reliability as the application may require.

3.12. Management Support Simulation Software

Fig. 3.39 shows the complete flow diagram for the Putnam Estimation model, and is formed by integrating the earlier sub-models into one comprehensive model. The complete model is partitioned into the Static and Dynamic sub-model areas as shown. Data from the lower (static) part of the flow model is used to conduct a simulation in time over the software life-cycle in the upper (dynamic) flow model.

A software simulation package based on the simulation model including the neural network and decision support system has been developed and its architecture has been discussed in Appendix IV. An illustrative simulation example is also included in Appendix IV for the sake of clarity.
FIG. 3.39 COMPREHENSIVE SYSTEM DYNAMICS FLOW DIAGRAM.
3.13. Conclusions

The development of a system dynamics simulation model entails a complete understanding of the software development environment, and its associated constraints. Causal relationships bring increased insight into the transitive dependencies of different variables on each other, which are not obvious from without. The flow diagram clearly brings out the inherent dynamics of the software development process, which must be understood by any software project manager to be able to keep the project under control, within budgets, and quality standards. The simulation software emphasizes the major advantages of system dynamics as a tool for gaining insight on the behavior of complex software systems and establishes how automated systems can be exploited to their fullest extent by combining decision support systems with artificial neural networks in a typical simulation environment.