"In many ways, managing a large computer programming project is like managing any other large undertaking - in more ways than most programmers believe. But in many other ways it is different - in more ways than most professional managers expect."

F.P. Brooks, Jr.

The Mythical Man-month (Broo75)
6.1 Project Dynamics

We live in a dynamic world. The software development is also a dynamic process. Software requirements change, customers (users) demand new functions and capabilities, resources fluctuate. In this chapter we shall model this change process, generating new manpower - time pattern and determination of cost and schedule increase as a result of extra-effort requirements.

In fact, for controlling underway software projects we need a real-time process controller as in case of space missions. A project manager would not think of executing a project if he can't measure the achievements, compare it with pre-determined targets and then make corrections. We have a manpower trajectory - given by ILC equation (chapter 5). When we have real data this can be updated and converge to the true trajectory. This lets us compare resource consumption versus achievement to see if the software production rates and times are in agreement. This technique allows us to control the software process and make revised estimates of where we are heading.

Another perspective of project dynamics is that we can also model the requirements change process in real-time before it is flagged off, so that the decision makers can know how much
the change will cost over the life cycle and what its slippage consequences will be. This allows managers to play management games - 'what if' possibilities.

6.2 Dynamic ILC Equation

We can derive II order differential equation from (5.5):

\[ \dot{\phi} = \frac{\tau}{t^2} \cdot \tau \cdot e^{-t^2/2\tau^2} \]

Rewriting -

\[ \dot{\phi} = \frac{\tau}{t^2} \cdot \frac{\tau}{t^2} \cdot e^{-t^2/2\tau^2} \]

\[ = \frac{\tau}{t^2} \left[ \bar{\phi} - \dot{\phi} \left(1 - e^{-t^2/2\tau^2}\right)\right] \]

\[ = \frac{\tau}{t^2} \left[ \bar{\phi} - \dot{\phi} \right] = f(t). (\bar{\phi} - \dot{\phi}) \]

This is well-known Diffusion Equation (MaSh77). For software project systems it signifies that the rate of accomplishment is proportional to the 'pace function' (which depends on the factors like achievements, software organization, software tools, state of technology being applied, customer requirements, specifications, pace setters, idea generators etc.) and the objective remaining to be accomplished (objectivity).

\[ \dot{\phi} = \frac{1}{t^2} . t. (\bar{\phi} - \dot{\phi}) \]

Differentiating -

\[ \ddot{\phi} = \frac{1}{t^2} \left[ \bar{\phi} - \dot{\phi} - t \dot{\phi} \right] \]
Rearranging terms we get -

\[
\dot{\phi} + \frac{1}{\tau^2} \phi + \frac{1}{\tau^2} \dot{\phi} = \frac{1}{\tau^2} = 8
\] ........ (6.2)

This is a non-linear II order differential equation encountered in electrical and mechanical systems except that \(8\) is constant and not a function of time and the coefficient of \(\phi'\) is a variable. (6.2) may be written as:

\[
\ddot{\phi} + \varepsilon(t)\dot{\phi} + \alpha\phi = \alpha\ddot{\phi}
\] ........ (6.3)

where \(\alpha = \frac{1}{\tau^2}\) is the shape factor which is the measure of the stiffness of the system;

and \(\varepsilon = \alpha t\) is the entropy which is the measure of work being grounded or losses in production due to factors responsible for confusion, indecision, frictional forces like strikes, non-cooperation etc.

6.3 Analogous Behaviour

A problem is better understood if it can be associated with some familiar phenomenon. Therefore, we always search for analogies when studying a new problem. In the study of an abstract process, similarities are very helpful, particularly if this can be shown to be analogous to some concrete phenomenon. It is then easy to gain some insight into the new problem from the knowledge of corresponding process.

6.3.1 Dynamic Physical Model of Software Process

Dynamic physical model of software development has an analogy between the system being studied and some other system of a different nature. The analogy depends on an underlying

87
similarity in the forces governing the behaviour of the system.

(1) Mechanical System

Fig. 6.1a represents a mass that is subjected to an applied force \( F(t) \) varying with time, a spring whose force is proportional to its extension or contraction, and a shock absorber that exerts a damping force proportional to the velocity of the mass. The system may represent, for example, the suspension of an automobile wheel when the automobile body is assumed to be immobile in a vertical direction. It can be shown that the motion of the system is described by the following differential equation:

\[
M \ddot{y} + D \dot{y} + K y = K F(t) \quad \ldots \ldots \quad (6.4)
\]

where  
\( y = \text{Distance moved} \)
\( M = \text{Mass} \)
\( K = \text{Stiffness of the spring} \)
\( D = \text{Damping factor of the shock absorber} \).

(2) Electrical System

Fig. 6.1b represents an electrical system with an inductance \( L \), a resistance \( R \), and a capacitance \( C \) connected in series with a voltage source that varies in time according to the function \( V(t) \). If \( q \) is the charge on the capacitor, it can be shown that the behaviour of the circuit is governed by the following equation:

\[
L \dddot{q} + R \dot{q} + \frac{q}{C} = \frac{V(t)}{C} \quad \ldots \ldots \quad (6.5)
\]

Inspection of the equations (6.3), (6.4) and (6.5) shows that they have close similarity and there are marked
equivalences in the three systems as tabulated in Tab. 6.1.

<table>
<thead>
<tr>
<th>SNo.</th>
<th>Mechanical System</th>
<th>Electrical System</th>
<th>Software System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Displacement</td>
<td>Charge</td>
<td>q</td>
</tr>
<tr>
<td>2.</td>
<td>Velocity</td>
<td>Current</td>
<td>i=(q)</td>
</tr>
<tr>
<td>3.</td>
<td>Force</td>
<td>Voltage</td>
<td>V</td>
</tr>
<tr>
<td>4.</td>
<td>Mass</td>
<td>Inductance</td>
<td>L</td>
</tr>
<tr>
<td>5.</td>
<td>Damping factor</td>
<td>Resistance</td>
<td>R</td>
</tr>
<tr>
<td>6.</td>
<td>Spring Constant</td>
<td>1/Capacitance</td>
<td>1/C</td>
</tr>
</tbody>
</table>

Tab. 6.1 - Equivalences of the three systems

A software system can be represented by a diagram as shown in Fig. 6.1c. In fact, the software system behaves more like a low pass filter or a communication channel (Toma85).

6.4 Experimental Analogs

The mechanical system (MDK) and electrical system (RLC circuit) are the analogs of software system. The performance of a software development process can be studied with the help of these analogs. In practice, it is simpler to modify the electrical system than to change the mechanical system.

In fact, a software engineering laboratory may be set up to perform such type of experiments. If, for example, a project is bouncing too much with a particular set of parameters, the electrical analog model will demonstrate this fact by showing that the charge (and therefore, voltage) on the condenser oscillates excessively. To predict what effect a
change in management parameters will have on the achievement, it is only necessary to change the values of the resistance or condenser in the electrical circuit and observe the effect on the way the voltage varies.

The process may be further simulated on interactive graphic terminal and may be visualized pictorially.

6.5 Project Data

With the help of resource ceiling (\(\Phi\)) and expected development time (\(\tau\)), we can draw the project profile (early estimates). When the project starts, real data is collected and used to correct this profile. The easiest way to follow and track the dynamic behaviour of a software system is to plot manpower (which is proportional to effort rate) versus time. So this turns into a problem in time series analysis. The linear ILC equation (5.8) may be used to fit the actual manpower data to get a revised estimates of future resource consumption, effort, duration and manpower (linear regression).

6.6 Requirements Changes and Systems Growth

When the requirements increase, manpower curve gets inflated as shown in Fig. 6.4, the development time and schedule get extended. For studying the effects of changes in requirements (6.3) may be solved step-by-step using numerical techniques. The solution can be perturbed in time by changing complexity \(8 = \Phi/\tau^2\). Increase in complexity is introduced when the customer changes requirements/specification.
For a software house an estimator for $8$ may be computed depending on the system characteristics (commercial, scientific, real-time etc.). Normally 8 will depend on the following factors:

1. Number and size of modules ($x_1$)
2. Number and size of database/input files ($x_2$)
3. Number and size of output reports ($x_3$)
4. Any other factor ($x_4$)

then $8 = k_1x_1 + k_2x_2 + k_3x_3 + k_4x_4 \quad \ldots \ldots \quad (6.6)$

where $k$'s are constants and may be empirically estimated. Given a new value of $8$, we can continue the numerical solution (Runge-Kutta method) for that point in time and in this way study the schedule slippage and cost overrun.

6.7 Summary

The main objective of the investigation of software dynamics is to understand the nature of forces operating in a software development process in order to determine their influence on the stability or growth of the system. The outcome of the study suggests some re-organization or changes in policy, that can solve existing problem or guide developments away from potentially dangerous directions.
FIG. 6.2 – SYSTEM GROWTH WITH COMPLEXITY