CHAPTER 2

SOFTWARE PROJECT AND ITS MANAGEMENT*

"Sometimes life deals you a lousy hand of cards. Your job is to do the best with what you have and work hard to improve things as the game progresses."

Anonymous

*A modified version of this chapter was published in Proceedings of Annual Convention of Computer Society of India, (GuTo84).
2.1 Software Project

The term 'project' is generally used in a loose sense. In fact, a project is a base for connected tasks. It may be regarded as a set of cohesive activities. We shall deal with only applied research and large scale quantity manufacturing activities. Norden (Nord75) gives a substantative definition of a project as follows:

"A development project is a finite sequence of purposeful, temporally ordered activities, operating on a homogeneous set of problem elements(tasks), to meet a specified set of objectives representing an increment of technological advance."

A task is homogeneous if it is composed of elements, each of which has at least one technological interdependence point in common with another element during the life span. So the primary characteristics of a project is that of connectedness. Unconnected tasks do not constitute a project. Such tasks could be done in parallel, independently of each other.

Connected tasks interact with each other. For example, the starting of some task may depend upon the output of
a preceding task. The connected tasks may be visualised by PERT/CPM networks.

A software project deals with the process of development of computer programs. Each software project has its own characteristics - its size, duration, technical contents, environment and requirements - which significantly affect the nature of project organization and resources required for successful execution/completion.

2.2 Program, System and Product

Brooks (Broo75) mentioned that the total effort for a project depends on the anticipated utilization of its output. If computer programs are to be developed for internal use by the individuals, the effort required will be considerably less than if a program is to be released as 'software product' to be used by external users. A software product must be specifically designed, structuredly coded, thoroughly tested, nicely documented and well maintained. Similarly, the effort needed for production of several computer programs which function interactively as a system is greater than the effort required to produce stand-alone independent programs.

The 'product' and 'system' effects are depicted in Fig. 2.1. This is a useful picture for establishing the dimensions of a project. The effort increases by four folds as we move from a program to a program system or from a program system to a program product and in order to produce a software product effort required is sixty four times. The extra effort
required to transfer programs into products and system is not reflected in additional source code. So the productivity rate in developing an individual program is much higher than for system or product.

The profile of a software project - its duration, manpower requirements, cost and management parameters are dependent upon the nature of the product. For most part of our study, we shall be dealing with programming system (software) product - which is most complex and demanding type of project - a large, interactive, deliverable system of programs.

A small project team is able to keep most of the information necessary for interfacing and integrating the system in their heads, communication is straightforward and there is no need of sophisticated management methods. As the size of the project increases, staff size grows, the control and communication requirements increase exponentially and are not satisfied by informal procedures.

2.3 Project Management

For holding the Asian Games, the construction of Indra Prastha Stadium in Delhi was started in 1980. It was estimated to cost Rs. 16 crores and was to be one of the largest indoor stadium ever built but it was completed as scheduled and within budget (Chat83). Is this possible in case of Computer Software Project?

Software Projects are often delayed. Product delivered
is unreliable. It is expensive to maintain. The IBM OS Project which involved over 5,000 man-years was late by many years. The OS had many bugs and it surpassed the estimated budget (Broo75).

Why is the difference between the two projects?

Part of the answer is that it is easier for a Civil Engineer to visualize the complexity of a huge construction project than for a software engineer to see the complexity of a large software project.

2.3.1 The Peculiar Nature of Software

Software products are in some respects like tangible products of conventional engineering like bridges, buildings, machines etc. But there are important differences as well (Hoar75). They are like creating musical nodes in the form of a melody. Also, in conventional engineering there are well established areas like strength of materials, material science etc. which are the backbone and help in producing blue prints before the execution of project starts.

In fact, the software is a logical rather than a physical product. That is why, the costs are concentrated in development rather than production. There is negligible cost in producing copies of software once it is developed. Since software does not wear out, reliability is determined by logical features such as correctness and fault-tolerance.

The software life cycle is an important conceptual tool for understanding the nature of software development. The software engineers have identified seven distinct stages through which a
project passes. These are (1) requirements analysis, (2) specification, (3) design, (4) coding, (5) testing, (6) documentation and (7) operation & maintenance and are collectively called software life cycle. The pie chart in Fig. 2.2 shows roughly the amount of time each stage takes.

### 2.4 Software Project Management Problems

To control the entire software life cycle is the task of project management which is directly responsible for planning, organizing, staffing, directing and controlling the project.

The difference between a software project's success or failure is attributed to good or poor practices in software project management. The main problems (Fig. 2.3) encountered are as follows:

#### 2.4.1 Inadequate Planning

This is the root cause of the failure of most of the unsuccessful projects. According to Metzger (Metz81) of all the unsuccessful projects of which he had knowledge, roughly 50% failed because of poor planning. The poor planning leads to large amount of wasted effort and idle time because of tasks being unnecessarily performed, overdone, poorly synchronised or poorly interfaced.

#### 2.4.2 Poor Control

Even a good planning is useless if the control exercised is not up-to-date and used to manage the project. The key to effective control is to break up the development into a number of small measurable steps and then to rigorously audit the
satisfactory completion of those steps. But in practice it is quite difficult.

2.4.3 Improper Resource Estimation

Without a firm idea of how much time and effort a task should take, the manager is in a soup to exercise good control. Accurate estimation of delivery dates and costs for a software development project are the most critical tasks of management. Historically, these estimates have been extremely poor, largely because there was little prior experience from which to draw on a specific system and because there were few metrics. Delivery dates for completed software products were set arbitrarily, with little awareness of the time that it would take to design, build, and test a complex software product. Such unrealistic deadlines could not be met and efforts to do so led to poor quality software. Systematic testing, in particular, was often given less attention, so many errors were discovered by users after product release, often resulting in huge maintenance expenses.

2.4.4 Semi-skilled Staff

In software development organizations the pattern of promotion has been from programmer to analyst and then to successive levels of management. A person who is a good programmer is not necessarily a good analyst or a good engineer or manager. When that person is promoted to a job that involves interpersonal skills dealing with non-technical issues, the programmer does poorly. Similarly, persons with good managerial skills or problem solving abilities are frequently unable to
follow through with the task of constructing a correctly functioning program. To get good people for a particular task has always been a problem. Also, good people leave the organization in between the project.

2.4.5 Unsatisfactory Project Progress

A key advance of the phased approach to software development is that there must be milestones (work products) associated with each project phase. In absence of such products, which is normally the case, it is difficult for a manager to see how well each individual or the entire project team is progressing.

2.5 Guidelines for the Success of a Project

How is the success of a software project guaranteed? What are the basic principles to guarantee a reliable product developed within budget and time schedule?

In order to answer these questions, we present a set of five guidelines to assist the project managers in making the most efficient and effective use of their own capability and other resources.

2.5.1 Employ Perfect Planning Using Life Cycle Approach

This guidance must be stressed utmost. More time may be devoted to the planning and this must be done by quite experienced, skilled and responsible persons who must consider the success of the project their prestige issue. The planning should not be affected by factors like badly defined contract, poor problem specification, inexperienced management team and
political pressures. Following are the essential steps of a good software project plan:

- clearly defined project overview
- well-defined milestones
- clear-cut project and product plans
- perfect validation plan
- reliable operations and maintenance plans

The overall planning must be oriented towards life cycle and phased development. The iterations must be confined to successive stages only. There should not be multiphase iteration loops. This is shown in Fig.2.4. The multiphase iteration loops result in cost and schedule overruns, software redesign and delayed recognition of problems.

2.5.2 Have Clear Accountability Structure and Control

Each individual in the project team should have a clear and well-defined statement of the results for which he or his group are accountable and there should be a clear understanding that he would be suitably rewarded depending on how well he produces those results.

This requires an adequate visibility into each person's or group's project performance. In order to achieve this visibility it is necessary to break the software process down into a large number of well-defined steps and to organize the project around these steps in such a way that each individual can tell whether he is doing his job well or not. One of the best methodologies for this purpose is Unit Development Folder (UDF) and its
associated cover sheet prepared by Ingrassia (Ingr76). This is a structured mechanism for organising and collecting software development products as they become available.

2.5.3 **Create Modern Software Engineering Environment**

The management should create a modern SEE — software engineering environment in the organization. In the most general sense this environment includes the technical methods, management procedures, the computing equipment, the software tools to support developers and the physical workspace. An ideal environment enhances the productivity of staff and provides a set of manual and automated tools that support the process of software production. Some examples of SEE's are Unix (Bell Labs.), INTERLISP, Smalltalk (the Dynabook), APSE (Ada Programming Support Environment) etc..

The application of modern software techniques helps in making the development process more visible. It aids in getting errors out early which can be nipped in the bud. It also produces an understandable and maintainable code and makes subsequent phases easier.

The emphasis is on the use of modern high-level languages (FORTRAN V, PL/1, Pascal), Program Design Language (PDL), Chief Programmer Team using Top-down development, structured programming and walk-throughs. Apart from this, the software design methodologies like Problem Statement Language/Problem Statement Analyser (PSL/PSA), Structured Analysis and Design Technique (SADT), Software Requirements
Engineering Methodology (SREM) and Jackson Methodology are quite useful for the success of a project (Fair85).

2.5.4 **Perform Validation Throughout the Project**

From reliability point of view it is essential to get the errors out as early as possible. For this purpose continuous validation throughout the life cycle is a must.

One of the reasons for the failure of software projects is to defer the detection and correction of errors until late i.e. in the testing phase after the code has been developed. This strategy is fatal because the most of the errors have already been introduced before coding begins and the later an error is detected and corrected the more it tells upon the success of the project.

In most of the projects the errors in the early stages (incomplete problem statement, ambiguous specifications, inconsistent assumption etc.) out number the coding errors (semantics, computational accuracy, control transfer etc.). This is shown in Fig 2.5 in case of IBM OS Project.

Validation throughout the project forces us to expand each phase of the software development life cycle to include an explicit validation activity as shown in Fig. 2.6.

2.5.5 **Engage Skilled and Sufficient Staff**

The management must find appropriate ways to use people on different projects and must accommodate the variation in staffing levels. Within the developmental organization it is the responsibility of the management to determine how to structure the organization and what assignments to be given to
individuals. It should be realized that there is a wide range of variability among individuals with respect to productivity and there is wide variation of skills of individuals in performance of specific tasks.

An important point of this guideline is to employ better and fewer people. This is required to reduce the communication overhead. The number of communication paths of N individuals is $N(N-1)/2$ i.e. of the order of square of the number of individuals; this becomes unmanageable when the group grows to a large size (Fig. 2.7).

Although there are many strategies (Fair85) yet the one adopted by IBM is the Chief Programmer Team (CPT) (Bake72). This concept recognizes that programmers have different levels of competence, therefore, the most competent should do the major work, with others in supporting roles (Rodg86).

2.6 Summary

The Computer Software is now considered as a commodity or a product. The discipline of Software Engineering caters for the development of this product. Many software projects fail due to mismanagement. This is because of the complex and incompletely-understood nature of the software.

The main thrust of this chapter is on the problems associated with the management of software development projects and the essential guidance for overcoming them. These guidelines are applicable to medium and large software projects and may be adapted to smaller projects as well.
FIG. 2.1 - EFFORT FOR PROGRAM, SYSTEM AND PRODUC

Project No EIP CS. 44.15
FIG. 2.2 - EFFORTS DISTRIBUTION IN PRODUCT DEVELOPMENT

(a) APPARENT EFFORTS (COST-WISE)

(b) TRUE EFFORTS (TIME-WISE)

FIG. 2.3 - MANY WAYS IN WHICH PROJECT IS A SUCCESS OR FAILURE
FIG. 2.4 - PHASED LIFE-CYCLE OF A PROJECT
IBM OS PROJECT

FIG. 2.5 - MOST OF THE ERRORS ARE IN EARLY STAGES
Fig. 2.7 - Growth of Comn Problems with More People