UNIT - IV

S/W Project planning Objectives, Decomposition techniques : S/W Sizing, Problem-based estimation, Process based estimation, Cost Estimation Models : COCOMO Model. S/W Design : Objectives, Principles, Concepts. Design methodologies Data design, Architectural design, procedural design, Object oriented concepts.

Unit - IV

11

SOFTWARE CHARACTERISTICS

Objectives

1. The software problems
   - 1.1 Cost of Software
   - 1.2 Reliability
   - 1.3 Change and Rework
2. Software as a Process
   - 2.1 Predictability
   - 2.2 Support Testability and Maintainability
   - 2.3 Early Defect Removal and Prevention
   - 2.4 Process Improvement
3. Software as a Product

Introduction

The electronic computers evolved in the 1940s. Then, the challenge was the hardware and all the early efforts were focused on designing the hardware. Hardware was where most technical difficulties existed. But then slowly with the advent of new techniques, the problem subsided. With the availability of cheaper and powerful machines, higher level languages, and more user-friendly operating systems, the applications of computers grew rapidly. In addition, the nature of software engineering evolved from simple programming exercises to developing software systems, which were much larger in scope, and required great effort by many people.

The techniques for writing simple programs could not be scaled up for developing software systems, and the computing world found itself in the midst of a "software crisis". At that time, the term software engineering was coin; in the conferences sponsored by NATO Science Committee in Europe in 1960UDe IEEE Glossary of Software Engineering defines software engineering as: "The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; i.e., the application of engineering to software.

[Now computer systems are used in such diverse areas as business applications, scientific work, video games, air traffic control, etc. This increase in the use of computers in every field has led to an increase in the need for software dramatically.

Further more the complexity of these systems is also increasing - imagine the complexity of the software for aircraft control or a telephone network monitoring system. The complexity has grown at such a pace that one is not able to deal with it. Consequently, many years after the software crisis was first declared, one finds that it has not yet ended. Software engineering is the discipline whose goal is to deal with this problem.

Today, software takes on a dual role. It is a product, and at the same time, the vehicle for delivering a product. As a product, it delivers the computing potential embodied by computer hardware. Whether it resides within a cellular phone or operates inside a mainframe computer, software is an information transformer - producing, managing, acquiring, modifying, displaying, or transmitting information that can be as simple as a single bit or as complex as a multimedia simulation.

As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environment

In this chapter, first the major reasons for the software problem and the major problems that software engineering faces are discussed. And then to gain an understanding of the software, some of the characteristics of software - as a product and as a process are discussed.

1. The Software Problem :

Software is not merely a collection of computer programs. IEEE defines software as a collection of computer programs, procedures, rules, and associated documentation and data. Unlike a program, the programming system product is generally not used only by the author of the program but is used largely by other people other than the developers of the system.

It is well documented or aided by other means, to help other people use the program. In a program, the presence of bug is not usually a major problem because the author itself generally uses it; if the program crashes the author will fix the problem and start using it again. These programs are not designed with such issues as portability, usability and reliability in mind.

Software falls under the category of programming systems product. The user may be of different backgrounds and so it is generally provided with a proper user interface. The programs are thoroughly tested before operational use so that there are no bugs left behind. And as diverse people having diverse hardware environment use the product, portability is a key issue.

As a thumb rule, a programming systems product costs approximately ten times more than the corresponding program. The software industry is usually interested in these commercial software systems or packages falling under programming systems product. It is also sometimes called production or industrial quality software.

The software problems are discussed here in detail:

1.1. Cost of Software :

The production of software is a labour intensive activity. Over the past few decades the cost of hardware has consistently decreased. With the advent of newer

and faster processors the cost of computing power has decreased. Similarly, the cost per bit of memory decreased more than 50 fold in two decades. On the other hand, the cost of software is increasing. Building it and maintaining it are labour intensive activities, but delays in delivery can be very costly and any undetected problems may cause loss of performance and frustrate users.

Fig. 1 shows that software development and maintenance costs have increased in the last few decades while the hardware cost has decreased constantly during the same period.

The size of software is usually measured in terms of Delivered Lines of Code (DLOC) and the productivity is measured in terms of DLOC per person-month. The cost of developing software is generally measured in terms of person-months of effort spent in development.

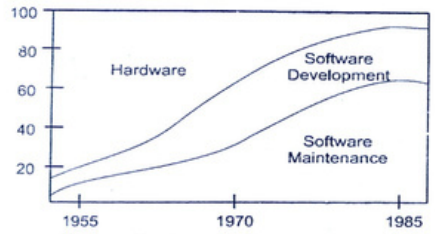


Fig. 1 : Hardware-software cost trends

The current productivity in the software industry is usually in the range of 300 to 1000 DLOC per person-month. And software companies charge the client for whom they are developing the software upwards of Rs. 1,00,000 per person-year or more than Rs. 8,000 per person-month Le., with the current productivity figures of the industry, Rs. 8 to Rs. 25 is charged per line of code. Moderately sized projects easily end up with software of 50,000 LOC. With this productivity, such software wili cost about Rs. 50 Million and Rs. 12.5 Million.

1.2. Reliability :

There are many Instances quoted about software projects that are behind schedule and have heavy cost overruns. The software industry has gained a reputation of not being able to deliver on time and within budget. For example, a Fortune 500 consumer products company plans to get an information system developed in nine months at the cost of Rs. 2,50,000 Two years later, after spending Rs. 2.5 million, the job was still not done, and it was estimated that another Rs. 3.6 million would be needed. The project was scrapped (because, the extra cost of Rs. 3.6 million was not worth the returns).

There are runaway projects in software industry. These are not projects that are somewhat late or over budget - it is one where the budget and schedule are out of control. The problem has become so severe that it has spawned an industry of its

own for which there are many consultancy firms that advice how to rein such projects.

The software industry is popular for not delivering software within schedule and budget and of producing software systems of poor quality. Many failures occur due to bugs that get introduced into the software, and as a result do what it is not supposed to do. For e.g. many banks have lost millions of rupees due to inaccuracies and other problems in their software. The software failure is very different from the failure of mechanical or electrical systems. These systems fail because of aging but software fails due to bugs or errors that get introduced during the design or development process. Hence, even though software may fail after operating correctly for some time, the bug that causes that failure was there from the start.

1.3. Change and Rework :

Once the software is installed and deployed, it enters the maintenance phase. This phase is usually divided into two types:

Corrective maintenance: This type of maintenance is needed to correct or debug some residual errors in the software as and when they are discovered leading to the software getting changed. Many of these errors surface only after the system has been in operation, sometimes for a long time.

Adaptive maintenance: Software must be often upgraded and enhanced to include more features and provide more services. One the system has been deployed; the environment in which it operates also changes. The changed software then changes the environment, which in turn requires further change. This phenomenon is called the law of software revolution. Maintenance due to this is adaptive maintenance.

Maintenance work is based on existing software as compared to development work that creates new software. So maintenance revolves around understanding existing software and maintainers spent most of their time trying to understand the software they have to modify. To test whether those aspects of the system that are not supposed to be modified are operating as they were before modification, regression testing is done. It involves executing old test cases to test that no new errors have been introduced.

Thus, maintenance involves understanding the existing software (code and related documents), understanding the effects of change, making the changes - to both the code and the documents - testing the new parts (changes), and retesting the old parts that were not changed.

One of the biggest problems in software development, particularly for large and complex systems, is that the requirements are not understood. The software does what it is not supposed to do. The requirements are "frozen" when it is believed that they are in good shape, and then the development proceeds. But as software is developed the client gets a better understanding of the system and new requirements are discovered which were not specified earlier.

This leads to rework; the requirements, the design, the code all have to be changed to accommodate new requirements. It is estimated that rework costs are 30 to 40% of the development cost.

2. Software as a Process :

A software process is a set of activities, together with ordering constraints among them, such that if the activities are performed properly and in accordance with the ordering constraints, the desired result is produced. The desired result is, high-quality software at low cost. Clearly, a process that does not scale up (i.e., cannot handle large software projects) or cannot produce good-quality software is not a suitable process.

The fundamental objectives of a process are optimality and scalability. Optimality means that the process should be able to produce high-quality software at low cost, and scalability means that it should also be applicable for large software projects. To achieve these objectives, a process should have some properties. Some of the important ones are discussed here.

2.1 Predictability :

Predictability of a process determines how accurately the outcome of following a process in a project can be predicted before the project is completed. If it is not predictable, the process is of limited use. If the past experiences to control costs and ensure quality are used, a process that is predictable must be used. With low predictability, the experience gained through projects is of little value. A predictable process is also said to be under statistical control.

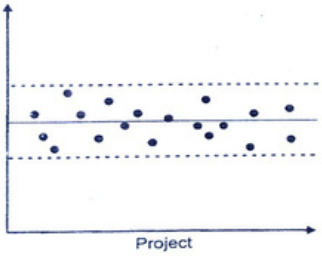


Fig. 2 : Process under statistical control

A process is under statistical control if following the same process produces similar results. This is show in Figure 2; the y-axis represents some property of interest (quality, productivity, etc.), and x-axis represents the projects. The dark line is the expected value of the property for this process. Statistical control implies that most projects will be within bound around the expected value.

Statistical control also implies that the value of the property should remain within a particular bound, if the same process is followed. If 20 errors per 100 LOC have been detected in the past for a process, then it is expected that with a high probability, this is the range of errors that will be detected during testing in future projects.

2.2. Support Testability and Maintainability :

As seen earlier, maintenance costs generally exceed the development costs during the life of the software. Thus, to reduce the overall cost of software, the goal of development should be to reduce the maintenance effort. According to a survey done by Bell Labs, the effort distribution within phases of a software process is as shown in Table 1. And similarly, the distribution of how programmers spend their time on different activities is shown in Table 1.

Requirements	10%
Design	20%
Coding	20%
Testing	50%

Table 1 : Effort Distribution with different phases

Writing Programs	13%
Reading Programs and manuals	16%
Job communication	32%
Others (including personal)	39%

Table 2 : How programmers spend their time

The Figures shows that most of the effort is spent in testing of software while only a small part is spent in programming. So, the goal of the process should not be to reduce the effort of design and coding, but to reduce the cost of testing and maintenance. Both testing and maintenance depend heavily on the design and coding of the software, and these costs can be considerably reduced if their Software is designed and coded to make testing and maintenance easier. Hence, during the early phases of the development process the prime issues should be "can it be easily tested" and "can it be easily modified"

2.3. Early Defect Removal and Prevention :

Errors can occur at any stage of the development cycle. An example distribution of error occurrences by phase is:

Requirement Analysis	20%
Design	30%
Coding	50%

Table 3 : Distribution of Defect Removal

As can be seen, errors occur throughout the development process. But the cost of correcting the errors of different phases is not the same and depends on when the error is detected and corrected. The relative cost of correcting errors originated in requirement phase as a function of where they are detected is shown in Figure 3.

As the Figure shows, an error that occurs during the requirements phase, if corrected during acceptance testing, can cost up to 100 times more than if it were corrected in the requirements phase itself. The reason is fairly obvious. The error in the requirements phase will affect the design and code.

And if the error were corrected after coding, both design and code would have to be changed, thereby increasing the cost of correction. So, one can deduce that errors should be detected in the same phase itself in which it has originated and not wait until testing.

2.4. Process Improvement :

A process is not a static entity. As the cost and quality of the software are dependent on the software process, it should be improved to satisfy goals as quality improvement and cost reduction. The process must learn from the previous experiences. Each project done using the existing process must feed information back to the process itself, which can then use this information for self-improvement.

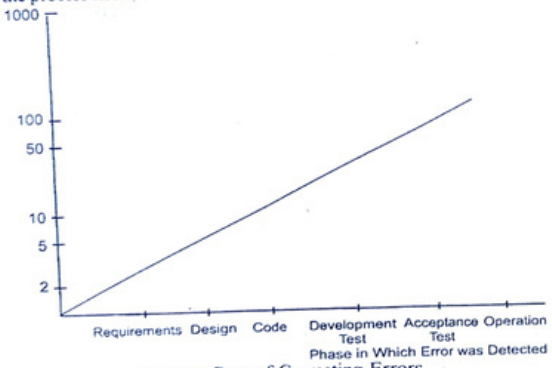


Fig. 3 : Cost of Correcting Errors

3. Software as a Product :

The goal of any engineering activity is to build something- a product. The civil engineer builds a dam, the aerospace engineer makes a plane, and the electrical engineer makes a circuit. The product of software engineering is a "software system". It is not as tangible as the other products, but it is a product nonetheless. It serves a function.

Software unlike other products is not a physical entity. One cannot touch or feel it to get an idea about its quality. Software is a logical entity and therefore, it is different from other engineered products.

To gain an understanding of the software, examine the characteristics of software that makes it different from .other things that human beings build.

Software is developed or engineered, it is not manufactured : Software does not automatically roll out of an assembly line. Sure, as one would learn later, there exists a number of tools for automating the process, especially for the generation of code, but development depends on the individual skills and creative abilities of developers. This ability is difficult to specify, difficult to quantify and even more difficult to standardize.

In most engineering disciplines, the manufacturing process is considered carefully because it determines the final cost of the product. Also, the process has to be managed closely to ensure that defects are not introduced. The same considerations apply to computer hardware products. For software, on the other hand, manufacturing is a trivial process of duplication. The software production process deals with design and implementation, rather than manufacturing.

Software is malleable : The characteristic that sets software apart from other engineering products is that software is malleable. The product itself can be modified (as opposed to its design) rather easily. This makes it quite different from other products as cars and ovens.

The malleability of software is often misused. While it is certainly possible to modify a plane or a bridge to satisfy some new needs - as to make it support more traffic but this is not as easy. This modification is not taken lightly and it is not directly attempted on the product itself; the design is modified and the impact of change is verified extensively. Software engineers are also often asked to make modifications in their system. Due to the malleability property, in practice it is not easy.

The code may be changed easily, but meeting the need for which the change was intended is not necessarily done so easily. One should indeed treat software like other engineering products in this regard: a change in software must be viewed as a change in the design rather than in the code, which is just an instance of the product.

The property of malleability of software can be used to advantage - provided it is done with a lot of discipline, this is where procedures and quality standards for making modifications become important.

Software does not "wear out" : The relationship between failure rate and time for hardware is shown in Figure 4. As can be seen, the figure is shaped as a bathtub and therefore often called as "bathtub curve". The relationship depicts that the hardware shows high failure rate in its early life cycle (these failures are due to manufacturing or design defects); defects are corrected, and the failure rate falls to a steady-state level for some period of time. As time passes by, the failure rate rises again as hardware begins to wear out due to cumulative affects of dust, vibration, abuse, temperature extremes, and many other environmental maladies.

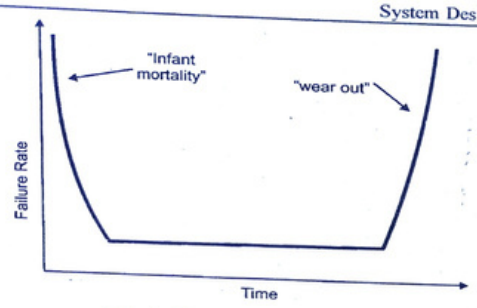


Fig. 4 :Hardware Failure Curve

But for software the curve is not as shown in Figure 4 because the software is not susceptible to environmental maladies. In theory, the curve for software failure against time should be as shown in Figure 5. The defects in the early stages will cause high failure rates in the early life cycle but as the defects are corrected, the failure rate drops and the curve flattens as shown in figure. But this is the over simplification of the actual failure rate of the software.

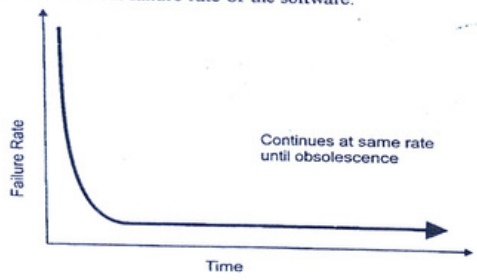


Fig. 5 :Idealized software Failure Curves

The actual failure rate for software is as shown in Figure 6. This contradiction from the earlier curve is because, during its life, the software undergoes maintenance. As changes are made, some new defects are introduced, causing the failure rate to spike and as the defects are corrected the failure rate again drops. But before the curve can return to the original steady-state, a new change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise- the software is deteriorating due to change.

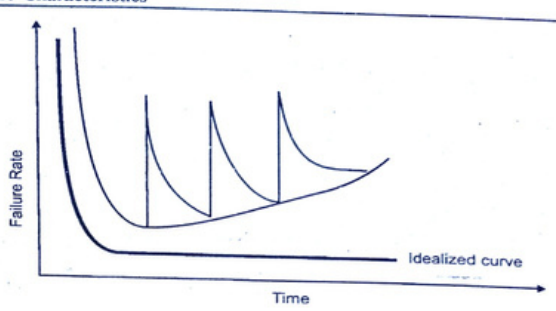


Fig. 6 : Actual Software Failure Curve

Another aspect of wear illustrates the difference between hardware and software. When a hardware component wears out, the part is replaced by a spare part. But no such spare parts exist for software. Therefore, the maintenance of software is much more difficult than the maintenance of hardware.

Most Software is Custom-Built, rather than being Assembled from Existing Components : If one looks at most of the other engineered products, one will find that in manufacturing these, a design is first made identifying the various components that go into each. These are then put together as per the original design. This approach affords an enormous amount of flexibility. Any number of people could be working independently in producing different components- so the manufacture of one-assembly could be entirely independent of another.

Now, if the organisation wants to increase production quantities or decrease the production cycle time, it can easily subcontract some of the jobs. So the final product is merely putting together several independently manufactured components. For example, for building hardware, the digital components are assembled so that proper function can be achieved. These digital components can be ordered 'off the self'.

Sadly, software engineers do not have this luxury. The software can be ordered off the shelf but as a complete unit, not as a software component that can be assembled into software programs. Fortunately, this situation is changing rapidly. The widespread use of object-oriented technology has resulted in the creation of software engineering components. Though the scenario is changing but this is only the beginning of this concept. This concept is usually known as "software reusability".

Questions

Short Questions:

1. What do you mean by cost of Software ?
2. Describe software reliability ?
3. What is predictability ?

Very short Questions

1. What do you mean by software problem ?
2. What is Maintainability of software ?

Long Question :

1. What are the software problem? Discuss in details.
2. Discuss the software characterstics.
3. Describe the characterstics of the software as a Process.
4. Describe the characterstics of the software as a Product.

❑❑❑

12

SOFTWARE PROJECT PLANING

Objectives

1. Why is Project Management Important?
2. Project Initiation
   - 2.1. The Need
   - 2.2. Scope of Work
   - 2.3. Feasibility Study
   - 2.4. Evaluation of Options
     - 2.4.1. Methodology
   - 2.5. Sizing and Effort Estimation
     - 2.5.1. Program Complexity Method
     - 2.5.2. Lines-of-code Method
     - 2.5.3. COCOMO Model
     - 2.5.4. Basic COCOMO
     - 2.5.5. Intermediate COCOMO
   - 2.6. Project Cost Estimates
     - 2.6.1. Manpower Cost
     - 2.6.2. Hardware Cost
     - 2.6.3. Software Cost :
     - 2.6.4. Travel Cost
     - 2.6.5. Training Cost
     - 2.6.6. Administration Cost
3. Risk Analysis
   - 3.1. Manpower Risk
   - 3.2. Technology Risk
   - 3.3. Customer/User Risk
   - 3.4. Environment Risk
4. Project Planning
   - 4.1 Methodology
   - 4.2. Risk
   - 4.3. Quality Plan

4.4. Configuration Management Plan
4.5. Project Scheduling
4.5.1. Identification of Activities
4.5.2. Allocation of Responsibilities
4.5.3. Scheduling of Activities
4.5.4. Identifying Milestones
4.5.5. Pictorial Descriptions
4.5.6. The Critical Path
5. Resource Plan
5.1. Hardware
5.2. Software
5.3. Tools
5.4. Communication
6. Project Tracking and Oversight
6.1 Schedule Tracking
6.2. Resource Tracking
6.3. Cost Tracking
6.4. Management Oversight
7. Project Metrics
7.1 Schedule & Effort Metrics
7.2. Quality Metrics
7.3 Cost Metrics
8. Project Closure
9. Miscellaneous Items
9.1 People Management
9.2 Resource Management
9.3. Quality v/s Schedule
9.4. Systems Integration Issues
9.5. Idealism v/s Realism
9.6. Sub-Contractor Management

1. Why is Project Management Important?

Project management has become an absolute necessity to ensure the success of any software project. For any project, project management is the process of directing the development of an acceptable system at a minimum cost within a specified time frame. Project mismanagement can deter or render ineffective the very best of analysis and design methods. The main results of mismanaged projects are:

— Unfulfilled or unidentified requirement or needs of the clients
— Cost over runs
— Late delivery
— Each phase and activity of a project needs to be monitored and managed.

Project management is a technique used by a manager to ensure successful completion of a project. It includes the following functions:

- Estimating resource requirements
- Scheduling tasks and events
- Providing for training and site preparation
- Selecting qualified staff and supervising their work
- Monitoring the project's program
- Documenting
- Periodic evaluating
- Contingency planning

From these functions, we can see that project management is a specialized area. Software Project Management is a very vast area and covers the management function of planning, organizing, staffing, leading and controlling software projects. Broadly, the following is what each of these entails:

Planning: Predetermining the course of action for the project to be able to achieve the specified objectives.

Organizing: Itemizing the project activities and then grouping these activities logically. It also involves assigning authorities and responsibilities for each of these tasks.

Staffing: Selecting appropriate personnel required to do the tasks as per the plan.

Leading: Creating an atmosphere that will assist and motivate people about their assignment tasks in the best manner possible.

Controlling: Ensuring that the project goes as per plan by measuring and correcting the performance activities.

It also uses tools and software packages for planning and managing each project. Managing projects also requires the following:

1. Top management commitment to setting project priorities and allocating resources to approved projects.

2. Active user participation to identify information needs, evaluate proposed improvements on a cost/benefit basis, provide committed resources and be receptive to training when scheduled.

3. A long-range plan that includes general project priorities, objectives, schedules and required resources.

Included in this is pressure from users who require the systems department to accept impractical tasks or deadlines. The result is rushed, compromised project, contrary to good system development practices. A further difficulty found in many organizations occur when individual departments acquire microcomputers without knowing about their requirements or consulting with the centralized computer facility. The result is often uncoordinated confusion that makes it difficult to plan or control projects.

Project Management is the most talked but also the most ignored area in any software project. The following key topics are covered under project management:

- Project Initiation - Project initiation activities covering the need for the system, Feasibility Study, risk associated with a software project, Evaluation of Options and Estimation methods. Under Evaluation of Options a quantitative weighted points method has been described to evaluate various options. Under Estimation, Program Complexity methods and Lines of Code method, the most commonly used methods are described in details with examples.

- Project Planning - This involves preparation of a number of Plans like Quality Plan, Configuration Management Plan, Contingency Plan etc. The key part of the Project Plan is Project Scheduling. This is described in detail by a step by step procedure giving details of Identification of Activities, Apportioning of Efforts, Setting dependencies between activities, Allocation of resources, setting start and end dates for each activities.

- Planning without Tracking is meaningless exercise. Techniques used for tracking and reviewing progress of projects is described under Time Sheets and review meetings.

- What you cannot measure, you cannot improve is an oft quoted statement. Project Management is no exception and the key metrics relating to schedule, quality and Costs.

- There are a number of practical realities relating to Project Management. These are discussed in a section titled Miscellaneous where People Management, Resources Management. Prioritization between Schedule and Quality, Idealism and Realism and Sub-Contractor Management are discussed.

2. Project Initiation :

Project initiation relates to a series of steps which must be done before any work is started on the project. These steps are essential in order to ensure that the project will provide the expected benefits and the requirement value for money. It also evaluates various options for doing the projects and choosing the best alternative. The project initiation comprises of the following steps:

- The Need
- Scope of Work
- Feasibility Study
- Evaluation of Options
- Sizing and Efforts Estimation
- Project Cost Estimates Risk Analysis

2.1. The Need :

For any project to start, there must first be need. There must be problem to solve or an efficiency to be improved or a bug to be fixed or a system to enhance. Without a need, a project cannot start. The need may come from the users of the

system, from the people who are managing the system or from the author of the system. Users have requirements from new systems to meet new business requirements or enhancements to the existing systems for launching new products which are adaptations of the existing products. System Manager will have needs to improve the performance of the system while the designer of the system will have to weed out the errors in the system and improve performance in the bottlenecks areas.

Once the need for the fresh system development or modification to an existing system has been identified, the need must be ratified. Ratification of the need is not required for fixing bugs as it is to be done on as-soon-as-possible basic. For all new requirements, it is a good practice to get the need checked by a second person. This would help in the following ways:

- to ensure that the need is really present
- to verify that the need cannot be met with existing systems
- to evaluate the cost-effectiveness of the need
- to confirm that all other options than modification to system are closed

The verification of need must be done by a senior person who is conversant with the business requirements and can take decisions on whether the change is required or not. Every change must have a documented list of benefits of the proposed change and must also list out the priority from a business need.

The need must be stated clearly as it may lead to misunderstanding at a later stage. The lead must state the following:

- Overview of requirement
- Interfaces with other products or systems
- Details of requirements
- Need for change in existing data
- Benefits of the system
- Priority for the system
- Initiator and approver of the requirement

Once the need has been identified, the technical personnel must evaluate the scope of work for the project.

2.2. Scope of Work :

Once the requirement has been received by software personnel, the first step is to check if the requirement is stated in clear terms and all details required to identify the scope of work are available in the specifications. The scope of work primarily defines the system boundaries. The Requirements Specification Document is prepared at this stage.

This document gives an overview of the system boundaries and other impacted areas. On completion of this document, it must be circulated to all impacted parties. This will enable all concerned to be aware of the change and ensure that they are given time to comment on the requirement and proposed changes. A presentation

of requirement produces much better result as people generally don't like to go through thick specifications. The main objective of this document is to create awareness amongst all concerned and get their views if something critical has been missed out.

2.3. Feasibility Study :

This is the most appropriate stage to do feasibility study for the system as the requirements definitions are available at an overview level and not too much efforts are spent on the project, so if the project is impractical, then this is right stage to point out if the system is feasible or not. The feasibility study is already covered in the previous chapter.

At the end of the feasibility study, if the system is found infeasible, then the project terminates at this stage, otherwise continues further with the next steps.

2.4. Evaluation of Options :

Once feasibility is established, then one has to evaluate all options and choose the one which best meets the requirements in a cost-effective and timely manner. The most common method for doing evaluation of options is a Weighted Point Analysis, which is described below.

One of the critical decisions an IT Manager has to make when approach by a user department for automation of some business areas is whether to go for a ready-made package or build a new customized solution. If the ready-made package solution is selected, should the package be used on 'as-is-where-is' basis or should the package be fine-tuned for the departments' needs. There can be argument for and against all the options. While packaged solutions offer tested and quick solutions, they have problems in interfacing the existing systems, meeting complete customer needs and enhancement is a problem as the organization will be dependent on the vendor technical know-how. On the other hand, customized developed will meet needs totally, bridge easily with existing systems but has the drawback of long lead times and untested code. How does an IT Manager take a decision under these circumstances?

This technique will enable the IT Manager to take an objective view of the situation and recommend the best solution to the user department. The benefits of this technique are as follows:

Objective Decision : The decision to make or buy has a lot of intangibles. This study will enable client to take an object decision.

Evaluation of Options :The client will be aware of all the options and its pros and cons.

Structured Analysis : The customer will be able to clarify his own requirements as a structured analysis will be done in this study.

The ultimate deliverable at the end of this exercise will be quantitative evaluation of various options, listing it's pros and cons and a recommendation of the best options.

2.4..1 Methodology :

The technique will be executed in the following stages:

- Requirement stage
- Questionnaire stage
- Ranking stage
- Finalization stage

Requirement Stage : This stage must be executed in two stage:

— Assimilation step
— Verification step

- **Assimilation Step :** In this step, the functional requirements of the area to be automated are to be accumulated by meeting functional heads and operational staff. The emphasis should be on gathering all salient features of the desired system rather than getting the details - in other words, have more breadth of coverage rather than depth. Apart from functional features, performance requirements, security, audit, recovery and contingency requirements of the proposed system should be noted. The preferred choice of hardware and the environment software should also be noted. The duration of this step will depend upon the number of functional heads to be met.

Verification Step : In this step our understanding of the customer's needs are confirmed by the software professional telling each functional head our understanding of client's requirements. This step will iron out all inconsistencies or conflicting information, which has been collected. At the end of this step, the software professional should have enough information to prepare the questionnaire. This step should take the same duration as the pervious step.

Questionnaire Stage : This stage will be executed in the following steps: Preparation Step

— Preparation Step
— Weightage Step
— Circulation Step
— Finalization Step

Preparation Step : In this step, the information gathered in the previous stage should be consolidated in the form of a questionnaire. The questionnaire must be classified into various section based on functionality, security, recovery, audit and contingency requirement. Additionally, there should be a summary questionnaire and a detailed questionnaire. The objective of the summary questionnaire is to shortlist products for the detailed evaluation. The detailed questionnaire needs to be administered on shortlisted products only. Items such as vendor support, experience, pricing should be part of the questionnaire.

Weightage Step : After the questionnaire id finalized, we must assign weightages to all the questions. Weightages should be assigned for summary and detailed questionnaire. Some guidelines for assigning weightage are as follows:

- Critical features or absolutely essential requirement - weightage of 5
- Important feature but client can manage if some automated work around is available - weightage of 4
- Essential feature but client can manage with some manual work around - weightage of 3
- Desirable features which client needs sometime in future - weightage of 2
- Nice to have features which are cosmetics in nature and does not affect the business needs (like colors or help features) - weightage of 1

Circulation Step - The next step is to get consensus on the questionnaire. It must be circulated to all function heads. After getting their comments, the revised questionnaire should be presented in a meeting with all function heads. This will ensure that all function heads are in concurrence and our understanding of requirements is correct.

Finalization Step - After getting all the feedback from the client the final version of the questionnaire must be prepared. At this stage a list of potential supplier of products with required functionality should be made. We should write a letter to all potential suppliers seeking product information. This information can also be sought at the start of this stage so that required information is available at the end of this stage.

Ranking Stage : This stage will have following steps:

— Shortlisting step
— Evaluation step
— Calculation step

Shortlisting Step - In this, the summary questionnaire is filled up for all vendors who have responded to the letter. After filling up the questionnaire and multiplying by the weightages, an overall score is assigned to each product. There will be a cutoff score and only products above the cutoff score should be taken up for detailed evaluation.

Evaluation Step - Appointments with each shortlisted vendor for the detailed evaluation should be sought. The detailed evaluation should have a presentation by the vendor about the company and the product to be followed by demonstration of the product. Based on the information presented and demonstrated, the detailed questionnaire should be filled up, at the vendor place so that we can get any clarifications if required. References checking can be done for any information. A three point scoring scale as given below can be used for scoring the features available in the product.

Score of 0, if function is not available and cannot be made available.

Score of 2, if function is a work around or functionality can be provided in the future.

Score of 4, if functionality is available immediately.

Calculation Step - After all product evaluation have been completed, the overall score for each option must be arrived by calculating the weighted average. A summary sheet comparing all products is prepared. This will enable us to know the best options which needs. After fine tuning the scores, all options, which meet a minimum of 70% requirement, must be included in the final report. If no option meets 70% of the requirement, then the recommendation must be to go in for customized software development.

Finalization Stage : A final version of the report should be prepared and packaged. This report along with a soft copy should be handed over to the customer. A presentation of evaluation to the customer will help in clearing any questions.

A typical final evaluation will appear as given below. The number of questions in the questionnaire have been reduced to keep it simple as we are only trying to understand a concept.

S.N.	Description	Weightage	Option1	Option2	Option3
1.	Security				
1.1	Password Control	4	0	8	8
1.2	Authorization Features	3	0	6	6
1.3	Limits Checking	2	0	4	4
2.	Application				
2.1	Maintenance of Account Codes	5	20	20	10
2.2	Alphanumeric Account Codes	3	6	24	6
2.3	Flexible Coding Structure	2	0	0	4
2.4	Posting of Entries	5	20	20	10
2.5	Tracking by department and sub-department	4	0	8	8
2.6	Multi-account and department posting in one voucher	4	0	16	8
2.7	Budget Comparison and Variance Analysis	3	0	0	6
2.8	Cost Allocation across departments	3	0	0	6
2.9	Flexible reports including B/S and P/L	5	10	10	10
2.10	What is Analysis	2	0	4	4

No	Description	Weightage	Option1	Option2	Option3
2.11	Multi-location and department consolidation	3	0	0	6
2.12	Receivables management including aging	4	8	0	8
2.13	Payables management and printing of cheques	4	8	0	8
3.	Recovery				
3.1	Ability to recover data	4	0	0	8
3.2	Recovery Technique-Environmental Software or Application Software	3	0	0	6
3.3	Reconciliation and rebuild utilities	3	0	6	8
4.	Support				
4.1	Training to Users	5	20	20	10
4.2	Telephone and E-mail Support	5	10	10	10
4.3	New Releases	4	8	8	8
5.	Miscellaneous items				
5.1	Warranty Period	3	12	12	12
5.2	Price	3	6	6	12
5.3	Trial period	2	4	4	4
	Total	**88**	**132**	**186**	**188**

From the above it is apparent that Option 2 and Option 3 are very close contenders. They must be evaluated closely from a cost-benefit angle before a final decision is taken but option 1 does not seem to be a good option at all. It must be noted the option 2 and option 3 work out to 53 percent (186/88*4) or 54 percent (188/88*4) of requirements. Thus both these options will also involve a fair amount of work before they can fully live up to expectations. But you are now sure that you have chosen the best possible alternative through an objective technique.

2.5. Sizing and Effort Estimation :

The assumption at this stage is you have zeroed in one of the options. The next step is to size the work and then estimate the effort. It is important that these activities are done in proper sequence sizing first and then effort estimation. The most common mistake is to go in the reverse order. The accuracy of a software project estimate is predicted on a number of things:

- the degree to which the planner has properly estimated the size of the product to be built
- the ability to translate the size estimate into human effort, calendar time and dollars (a function of the availability of reliable software metrics form past projects)
- the degree to which the project plan reflects the abilities of the software team, and
- the stability of product requirements and the environment that supports the software engineering effort.

In the context of project planning, size refers to a quantifiable outcome of the software project. If a direct approach is taken, size can be measured in lines of code (LOC). If an indirect approach is chosen, size is represented as function point (FP).

Sizing is estimating the dimensions of scope of work in terms of deliverables-it could be in terms of function points or lines of code or any other unit of measure which is used by the organization for dimensioning software projects. This size is then converted into effort using productivity norms like number of function points or lines of code produced per day. A number of standard procedures used by software engineers for sizing and effort estimation are given below. You will then have to choose the method that is best suited for your environment.

A very good technique used by many organizations to ensure that estimation is done properly is to use a Delphi technique. You ask a set of experts (say 5) to estimate the effort by a methodology of their choice. The estimates for the project could be quite different depending upon the different assumptions made by the estimator. After all the estimates have been rationalized so that we can compare now, ignore the two extreme estimates the least and the highest and take an average of the middle three to arrive at the final estimate for the project. This technique is particularly useful as there is no sure-shot method of estimation for a software project and this method has the averaging effect of many software professionals and is considered a useful exercise by many organizations, specially when they are bidding mega proposals.

2.5.1. Program Complexity Method :

Under this method, the primary assumption is that enough details of requirements are available to make a list of programs required for development of the system. If this information cannot be estimated from the requirements, this estimation methodology will not work. The list of programs are then classified into 3 to 5 categories as Very Complex, Complex, Medium Complex, Simple and Very Simple. There can be guidelines for classifying the programs into various categories. Typical guidelines could be:

- Any program which updates the database must be classified as Complex or Very Complex
- Any program which has a large number of calculations must be classed as Very Complex

• Any program which only involves retrieval of data must be classified as Simple or Very Simple

• Any program which involves communication with customer or regulatory interface must have a minimum of Medium Complexity

Based on some guidelines as stated above, programs must be classified into categories. If we attach a weightage to each classification, we can arrive at a numeric value of the complexity of the system. In other words, we have sized the complexity of the system. Let us see how this works with an example. Let us assume that you are trying to develop a Financial Accounting package with basic functionality. The complexity of the system could be calculated as shown below.

List of programs	Very Complex	Complex	Medium	Simple	Very Simple
Maintenance programs		5	3		
Processing programs	5	3	2		
Batch programs	3	2			
Reports			4	5	10
Queries				5	12
Security			2	3	4
Total	**8**	**10**	**11**	**13**	**26**

If you give a 5-4-3-2-1 weightage for the above categories the total complexity of the system works out to

$8 \times 5 + 10 \times 4 + 11 \times 3 + 13 \times 2 + 26 \times 1$ - : 145 complexity points.

The number given in each row are the number of programs in each category for that module. If systems are evaluated across common guidelines, the complexity points could be used to compare systems for complexity and thus appropriate manpower can be assigned for the project. It must be noted that we have just completed the sizing of the project. The number of person-days for completing the programs (coding and unit testing only) under each category need not be in the ratio of 1 :2:3:4:5. This will depend on the skills of the people in the organization, the maturity level of the organization and a number of other subjective factors. Let us for this exercise that the number of person days to complete coding and unit testing for programs is as given below:

Very Complex	:	20 Person days
Complex	:	13 Person days
Medium	:	8 Person days
Simple	:	5 Person days
Very Simple	:	3 Person days

Using the above assumptions, the total effort for the project works out as shown below :

List of programs	Very Complex	Complex	Medium	Simple	Very Simple
Maintenance programs		65	24		
Processing programs	100	39	16		
Batch programs	60	26			
Reports			32	25	30
Queries				25	36
Security			16	16	12
Total	**160**	**130**	**88**	**65**	**78**

Thus the total effort for the project works out as given below:

160 + 130 + 88 + 65 + 78 = 521 person days.

This translates into 24 person month or 2 person year project assuming that there are 22 working days in a month. The effort which is calculated above only relates to coding and unit testing phase of the project when the team will have it's peak size. We now need to calculate the effort for the other phases of the project. Typically, a project following the waterfall life cycle model goes through the following phases:

- Initiation
- Analysis
- Design
- Coding and Unit testing
- System Integrated testing
- User Acceptance testing
- Implementation
- Post Implementation Support
- Project Management
- Documentation

The effort for the above phases can be calculated as a percentage of the effort estimated for coding and unit testing as shown below.

Stages of Project	Percentage of Coding	Percentage of Project
Initiation	16.7	5
Analysis	50	15
Design	50	15
Coding and Unit Testing	100	30
System Integrated Testing	33.3	10
User Acceptance Testing	33.3	10
Implementation	16.7	5
Project Management	16.7	5
Documentation	16.7	5
Total		**100**

Based on the above table the total effort estimate for the entire project will work out as shown below.

Stages of Project	Percentage of Coding	Percentage of Project
Initiation	16.7	87
Analysis	50	260
Design	50	260
Coding and Unit Testing	100	521
System Integrated Testing	33.3	174
User Acceptance Testing	33.3	174
Implementation	16.7	87
Project Management	16.7	87
Documentation	16.7	87
Total		**1737**

As you can see the total effort for the project suddenly balloons up by about 200 percent to become a 79 person month or 6.5 person year project. The percentage used for the estimation are guidelines and should be fine tuned based on the ground realities of the project. If personnel who are going to do the analysis are familiar with the application, then the time taken for analysis will be much lesser. If the requirement is an enhancement to an existing system, the design effort may take lesser or more effort depending on the quality of the design of the existing system. After making the adjustment for ground realities, it is normal to provide a buffer for items which were missed out or to provide for contingencies. The best indicator for contingency provision is the history or track record of organization. If the projects executed over the last 2 years have shown an average deviation of 23 percent between estimated effort and actual effort, then a 23 percent contingency provision must be provided. It must be noted that the contingency provision could also bring down the effort estimate if we find that all projects over the last two years have been showing a positive deviation between planned effort and actual effort.

Once the above exercise is completed, we can state that the estimation exercise is complete.

2.5.2. Lines-of-code Method :

The lines-of-code method for estimation is an adaptation of the Program Complexity method. The methodology followed is very similar to the technique given under Program Complexity method. This steps involved are as follows:

- Estimate the number of lines of code for each program
- Compute system size by adding the number of lines of code across all programs

- Convert lines of codes to effort using productivity figures
- Evaluate effort for other stages of project
- Make final adjustments to estimate

The above steps are explained with the same example described under Program Complexity method. A common way to estimate the lines of code is to calculate the expected value using the following formula:

$$\text{Expected Line of Code} = \frac{a + 4m + b}{6}$$

where a = Optimistic lines of code
m = Most likely lines of code
b = Pessimistic lines of code

Thus if we estimate a program will have 1800 line optimistically, most likely as 2400 and pessimistic as 2650, then the expected lines of code for the program will work out to be

$$\frac{1800 + 4 \times 2400 + 2650}{6} = 2340$$

We must calculate the expected lines of code for every program in the system and build a table as shown below. It must be emphasized here that the person estimating the optimistic, most likely and pessimistic lines of code must be a very experienced person with good knowledge of the application and the proposed environment on which the system is to be developed. It is this input which ultimately leads to the total effort for the project.

Once the lines of code for all programs have been estimated as stated above, the size of the project can be computed as shown below.

List of programs	Lines of Code
Maintenance programs	9,120
Processing programs	15,200
Batch programs	7,200
Reports	8,620
Queries	5,080
Security	5,300
Total	**50,520**

The size of the project has been estimated at 50,520 lines of code. The lines of code given in the above table are the expected lines of code calculated for each module as described above. Once the size has been estimated, this can be tracked by the organization. Let us assume that based on the data of projects for the last two years, it has been found that on an average 100 lines of unit tested code are produced

every working day across all types of programs (very complex to very simple) on the target environment for this project. If organizational figures are not available, industry figures on such statistics can be used for estimation. This average productivity figure will also take into consideration that some programmers are more productive than others. If this productivity figure is applied to the table above, we will get the effort estimate for coding and unit testing phase of the project as given below:

List of programs	Lines of Code	Effort
Maintenance programs	9,120	91
Processing programs	15,200	152
Batch programs	7,200	72
Reports	8,620	86
Queries	5,080	51
Security	5,300	53
Total	**50,520**	**505**

The total effort for coding and 'unit testing works out to 505 person days. The effort for the other phases of the project are estimated using the percentages as described under Program Complexity method. If the same percentages are used, the total effort for the project will be as shown below.

Stages of Project	Percentage of Coding and Unit Testing	Efforts in Person Days
Initiation	16.7	84
Analysis	50	253
Design	50	253
Coding and Unit Testing	100	505
System Integrated Testing	33.3	168
User Acceptance Testing	33.3	168
Implementation	16.7	84
Project Management	16.7	84
Documentation	16.7	84
Total		**1683**

effort depending on the quality of the design of the existing system. After making the adjustment for ground realities, it is normal to provide a buffer for items which were missed out or to provide for contingencies. The best indicator for contingency provision is the history or track record of organization. If the projects executed over the last 2 years have shown an average deviation of 23 percent between

estimated effort and actual effort, then a 23 percent contingency provision must be provided. It must be noted that the contingency provision could also bring down the effort estimate if we find that all projects over the last two years have been showing a positive deviation between planned effort and actual effort.

Once the above exercise is completed, we can state that the estimation exercise is complete.

2.5.3. COCOMO Model :

Barry Boehm introduced a hierarchy of software estimation models bearing the name COCOMO, for Constructive Cost Model Boehm's hierarchy of models takes the following form:

Model 1 : Basic COCOMO is a static single-valued model that computes software development effort (and cost) as a function of program size expressed in the estimated lines of code.

Model 2 : Intermediate COCOMO computes software development effort as a function of program size and a set of "cost drivers" that include subjective assessments of product, hardware, personnel and project attributes.

Model 3 : Advanced COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

To illustrate COCOMO, we present an overview of the basic and intermediate versions. COCOMO may be applied to three classes of software projects as given below :

- **Organic mode** - relatively small, simple software projects in which small teams with good application experience work to a set of less than rigid requirements (e.g. thermal analysis program developed for a heat transfer group)
- **Semi-detached mode** - an intermediate (in size and complexity) software project in which teams with mixed experience levels must meet a mix of rigid and less than rigid requirements (e.g. a transaction processing system with fixed requirements for terminal hardware and database software)
- **Embedded mode** - a software project that must be developed within a set of tight hardware, software ad operational constraints (e.g. flight control software for aircraft)

2.5.4. Basic COCOMO :

The basic COCOMO equations take the form:

$E = Ab(KLOC) \exp(Bb)$

$D = Cb(E) \exp(Db)$

Where E is the effort applied in person-months, D is the development time in chronological months (elapsed time), and KLOC is the estimated number of delivered lines of code for the project (expressed in thousands). The coefficient Ab and Cb and the exponents Bb and Db are given in table below.

Software Project	Ab	Bb	Cb	Db
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

The basic model is expanded to consider a set of "cost driver attributes" that can be grouped into four major categories:

- Product attributes
  Required software reliability
  Size of application database
  Complexity of the product
- Hardware attributes
  Run-time performance constraints
  Memory constraints
  Volatility of the virtual machine environment
  Required turnaround time
- Personnel attributes
  Analyst capability
  Software engineer capability
  Applications experience
  Virtual machine experience
  Programming language experience
- Project attributes
  Use of software tools
  Application of software engineering methods
  Required development schedule

Each of the 15 attributes listed above is rated on a 6-point scale that ranges from "very low" to "extra high" (in importance or value). Based on the rating, an effort multiplier is determined from tables published by Bochm and the product of all effort multipliers results is an effort Adjustment Factor (EAF).

2.5.5. Intermediate COCOMO :

The intermediate COCOMO equation takes the form:

$$E = Ai(LOC) \exp (Bi) \times EAF$$

Where E is the effort applied in person-months and LOC is the estimated number of delivered lines of code for the project. The coefficient Ai and the exponent Bi are given in table below.

Software Project	Ai	Bi
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.20

COCOMO represents a comprehensive empirical model for software estimation. Today, a software cost estimation model is doing well if it can estimate software development costs within 20% of the actual costs, 70% of the time, and on its own turf (that is, within the class of projects to which it has been calibrated)... This is not as precise as we might like, but it is accurate enough to provide a good deal of help in software engineering economic analysis and decision making.

To illustrate the use of COCOMO, we apply the basic model to the CAD software. Using the figures from the first table, we use the semidetached model to get

E = 3.0 (LOC) exp (1.12)
= 3.0 (33.3) 1 .12
= 152 person-months

To compute recommended project duration, we use the effort estimate described above.

D = 2.5 (E) exp (0.35)
= 2.5 (152) 0.35
= 14.5 months

The value for project duration enables the planner to determine a recommended number of people, N, for the project.

N = E/D
= 152/14.5
= 11 people

In reality, the planner may decide to use only four people and extend the project duration accordingly.

2.6. Project Cost Estimates :

You have so far estimated the effort in terms of person days for doing the project. This forms only one component of the total project. Let us now see how to estimate the total project cost. A typical software project comprises of the following expense heads :

2.6.1. Manpower Cost :

The total manpower effort for the project would have been finalized using one of the technique described earlier. For estimating manpower costs, it is normal practice to categorize the manpower into three to five categories and associate a monthly rate for each category. The rate should only include costs directly paid towards salaries, subcontracting fees and related perquisites.

2.6.2. Hardware Cost :

Hardware cost will also have to be computed on a unit costing basis. Let us take a typical project which uses personal computer as front-end workstations and

a Unix server at the backend. It is possible that the ~nix server is shared across many projects. The organization must work out unit costs for a month for each hardware item which can be utilized for a project. There could be a cost of Rs 3,000 per month for use of PC as a workstation. There could be a monthly unit cost for every MB of disk space used on the Unix server. If the system connects to external networks there could be a unit rate for dedicated ports or rate per hour of usage for shared ports. These rates must typically cover, depreciation costs, funding costs, insurance and annual maintenance charges. The challenge at this moment of time is to estimate the hardware and data communication equipment requirements for the project. These are relatively easy to determine once the total effort, team size and data volumes are known. The exact estimation technique may vary from machine to machine but the estimation is just as arithmetic of requirements and equipment required to meet the requirement. You cannot prescribe one formula for this calculation as it varies from environment to environment.

2.6.3. Software Cost :

Software cost are normally assigned on number of licenses or number of users using one license of the software. Again these are easy to determine once the environmental software is decided and the project team organization is finalized. The organization must have worked out a unit cost for every license of the software on a monthly basis. The unit cost of software normally includes depreciation, funding costs and annual maintenance charges.

2.6.4. Travel Cost :

Travel cost must be estimated based on the requirements for travel for the project. Typically, this could include costs of travel for review meetings, requirements specifications and clarifications, implementations etc. This must be done based on best available data at this stage.

2.6.5. Training Cost :

Training costs will include the cost of application, technical and general training which the project team members will have to undergo before they start working on the project. It could also include costs relating to travel for training, faculty charges and premises/equipment charges for training. This again has to be estimated based on the best available information.

2.6.6. Administration Cost :

Administration costs typically include charges relating to premises, utilities, communication and conveyance costs. It could also involve the costs of corporate overheads. These costs are normally worked out in most organizations on a per seat cost and the project is charged on the number of seats occupied it. These cost will have to be worked out based on the organizations policies for allocation of various costs.

3. Risk Analysis :

Software projects have a number of risks associated with them.

3.1 Manpower Risk :

This relates to rate of people leaving the project when the project is still in the development stage. This can cause substantial delays as new people may have to be recruited and trained which should lead to redoing some work as the incoming person may want to start things afresh rather than takeover the previous persons work and continue from there. The impact on the project is more severe if one of the senior members of the project suddenly leaves the project as the new person might want to make changes with higher impact on schedules and deliverables.

This risk can be addressed by either having a strong process driven organization or overstaffing the project to cater to such contingencies. An organization cannot be converted into a process driven organization overnight. It will have to be a long term strategy of the organization. A process driven organization will not feel the impact of the change in personnel as processes, by their very definition are supposed to be people independent. Thus, changes in personnel can be handled without major impact in organization with strong processes. In the field of software, there are very few organizations with such a degree of process orientation.

Another way of addressing the risk in a more common way is overstaffing. This will have cost impact but the impact on schedules may be prudent to have a backup person for critical positions so that change in personnel is eased out. For development personnel, it might suffice to have a pool of people who might be assigned to a low priority project and can be pulled into the critical project at short notice. This will help reduce costs as the pool could be common across many projects.

Each of the above risk ease technique has it's own pros and cons. The organization has to decide on the best route depending upon the criticality of the project and constraints of the organization.

3.2. Technology Risk :

Technology in today's world is becoming extremely complicated as what was the 'in' thing yesterday is the 'out' thing tomorrow. The shelf life of various technologies is between six months to two years in a rapidly changing world. There are two kinds of risks related to technology - using New technology and using Outdated technology. Both have their own set of risks.

If you are using new technology, the major risks are availability of manpower with skills in that area and stability of the product itself. These can be countered with effective training and a good acceptance test on the product but mere classroom training may not be sufficient for a person to program effectively using new tool. In such cases, a buffer period for people to play around with the tool and try out it's features, develop standards for using the tolls must be provided in the project schedule.

If you are using an outdated tool, the risk is manpower retention as people always want to work on the latest technology. They would like to move to organizations which are using the latest technology. This risk has to be addressed as described under manpower risk and a clear directive of the organization's plans to move to the latest version of the tool.

Another related risk is that the project starts on the latest technology but by the end of the project a new version of technology has come into the market and the project is implemented with obsolete technology. The best decision under the circumstances is to go ahead and implement the software with the older version of the tools. After successful implementation and stabilization of the product, a new project can be initiated to convert the software from the older version to the current version.

3.3. Customer/User Risk :

A software project is a joint exercise between customers and the software project team. A project is completed successfully if both work in synergy understanding the goals and working as a team. However, if there is a wall in between the two and lack of communication, the project may never see it's end. It has been observed that synergy between customers and software teams is higher where customers are computer literate than with customers who are absolutely new to computer systems. Customers who are computer literate understand the importance of specifying requirements upfront, implications of making changes during the course of the project and the improvement in quality of deliverables by testing the system with a structured test plan.

If customer's knowledge of software development is below acceptable level, it is imperative that they are trained on the software development life cycle, computer fundamentals and made aware of the various issues relating to software development. This will ease the risk to some extent.

3.4. Environment Risk :

This relates to risk depending on the location of the project. There are some countries where power supply is erratic. If a software project is being done in such a country, the risk has to be addressed by having appropriate UPS or generators. Government policies and rules made by regulators relating to imports, taxes and visas could have impact on software projects. The list of these risks can go endlessly. The key point to be noted is addressing such risk is to make sure that the project manager is aware of them and the action to be taken in case of the event occurring. It is impossible to plan for all these risks. Some of them are totally beyond anyone's control.

4. Project Planning :

After the cost has been approved for the project, the first step in the execution of the project is preparing the Project Plan. Project planning involves all aspects

relating to the project. It is a document which can be used as a reference to know how various activities are to be done, when they are to be done, their dependencies, the effort required for each activity. Typically Project Planning involves the following activities:

- Methodology
- Risk
- Quality Plan
- Configuration Management Plan
- Project Schedule
- Resource Plan

4.1 Methodology :

A methodology is nothing but a series of steps which need to be carried out to meet an end goal. In software development, there are number of methodologies which can be followed for development of a software product. Each of these methodologies have different variations and each has their own pros and cons. The most popular methodologies used for software development are:

- Waterfall Model
- Prototyping
- Rapid Application Development

We have already discussed this methodologies in the prior sections. The methodology selected for the project and the various key steps or milestones in the methodology must be described in the Project Plan.

4.2. Risk :

The typical risks associated with a project have already been explained earlier. In the Project Plan document, the risk associated with the project must be described and the steps taken to ease the risk must also be detailed. The risk must be described from a technology and business perspective. The objective of this section is that management is kept aware of the risks and it's consequences in the project.

4.3 Quality Plan :

A Quality Plan describes the various checks that will be done to improve the quality of deliverables. Typically this will contain the following details:

- Process for reviewing deliverables
- Testing process
- Walkthroughs
- Process Audits
- Adherence to Standards

4.4. Configuration Management Plan :

Configuration Management involves maintaining control on the versions of software. Typically, this will involve:

- Identification of Configuration Items
- Checking In-Out process
- Version Control of Software
- Reporting and Analysis

4.5. Project Scheduling :

The task of scheduling the project is the most difficult and important part of the Project Plan. It is an extremely complex task as you are dealing with various things which are probabilistic in nature. A list of activities or tasks is made for the project- these may expand or contract depending on changes which occur after the project has commenced. Allocation of responsibility to personnel is done for completing the activities without knowledge whether the people will stay till the end of the project and start dates and end dates are also assigned based on estimation and resources allocated to the activity, which again is highly probabilistic. One of the questions which may be asked at this stage is why have a project schedule when we are not certain of anything in the project schedule? The reason is that you want to know how you are doing at a given instant against the overall objective. A Project Schedule can thus be viewed as a road map to completing the project on schedule. A project schedule which is not being tracked is as useless as a car without a steering wheel. You know what all you have to do to get to the final destination but are unable to take corrective actions when you encounter roadblocks. Thus it is important not only to have a project schedule but also track against the schedule so that timely corrective actions can be taken.

Project Schedule comprises of the following steps:

- Identification of Activities
- Allocation of Responsibilities
- Scheduling of Activities
- Identifying Milestones
- Pictorial descriptions
- The Critical Path

4.5.1. Identification of Activities :

This step involves identifying the various steps or tasks to be performed in the project. The list of activities can be drawn up at an intermediate level. The important point is that the list of activities must be comprehensive so that no activity is missed out. It must act as a checklist to ensure that all tasks are completed before we say that the project is complete. Project Manager normally have a tendency to say that the project is complete as soon as testing has started - a list of activities with proper tracking will ensure that the management gets a clearer picture of the status.

The level of detail of activities for a project is important. It is ideal if the level of planning is done at the lowest level as shown next page :

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
Project Initiation					
User Request					
User Request Review					
Feasibility Study					
Estimation					
Approval of Estimates					
Risk Analysis					
Planning and Tracking					
Initial Plan					
Tracking and Review					
Revision of Plan					
Life Cycle Activities					
Functional Specifications					
Module 1 to Module					
Review of FS					
Design Specifications					
Module 1 to Module n					
Review of Design					
Development					
Program Specifications					
Program 1 to Program n					
Review of Specs					
Unit Test Plans					

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
Program 1 to Program n					
Review of UTPs					
Coding					
Program 1 to Program n					
Review of Code					
Unit Testing					
Program 1 to Program n					
System Test Plan					
Module 1 to Module n					
Review					
User Acceptance Plan					
Module 1 to Module n					
Review					
Documentation					
Module 1 to Module n					
Review					
System Testing					
User Acceptance Testing					
Packaging and Release					
Implementation					
Planning					
Testing Conversion					
Freezing data					
Database Definition					

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
Transfer to Production					
Conversion					
Parallel Run					
Live Operations					
Post Implement Support					

In the above project schedule, we have identified activities upto lowest level of detail - modules for Functional Specification and programs for Development. While this is the ideal plan, it might not be the most practical. You must be able to track activities at the identified level of detail. This becomes more and more complex as the number of activities become large. Imagine a situation where a system had 5 modules with an average of 50 programs in each module - you will have more than 1000 activities in your project plan. Tracking at that level of detail can be an extremely complex exercise. Tracking at that level of detail without a sophisticated project management tool will be a futile exercise. The idea is to have detailed level planning with summary consolidation for management reporting. If tracking at a detail level is not feasible, then it is better to plan the project at summary activities and not at detail level. A typical project planned at summary level will look as shown below.

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
Project Initiation					
User Request					
User Request Review					
Feasibility Study					
Estimation					
Approval of Estimates					
Risk Analysis					
Planning and Tracking					
Initial Plan					
Tracking and Review					

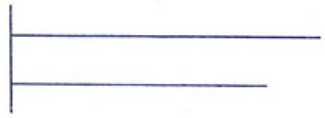
Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
Revision of Plan					
Life Cycle Activities					
Functional Specifications					
Design Specifications					
Development Unit Test Plans					
Coding					
Unit Testing					
System Test Plan					
User Acceptance Plan					
Documentation					
System Testing					
User Acceptance Testing					
Packaging and Release					
Implementation					
Planning					
Testing Conversion					
Freezing data					
Database Definition					
Transfer to Production					
Conversion					
Parallel Run					
Live Operations					
Post Implement Support					

As you can see from the above table, the module and program level details are removed from the schedule. There can be an intermediate level where planning for some items is done at module level and some at a summary of program level. The main criteria for deciding the level of detail of activities rs the ability to track

progress at the microscopic level. Planning without tracking is like designing a car without steering wheel - it will move but you do not know where you fire going nor are you able to control it.

Another factor which is important in deciding the level of activities for a project is dependencies. A project comprises of a number of activities or tasks. These tasks could be independent tasks or dependent tasks. Independent tasks are those tasks which cap start at any instant in the life cycle Irrespective of the status of other activities - User Acceptance Test cannot start unless System Test is Completed or Development cannot start till Design is completed. Activities must be identified such ,hat dependent activities are clearly defined as discrete activities and the dependencies are also stated explicitly. There may be many dependencies for an activity. It is not important to identify all dependencies, it is more critical to identify the important dependencies which have an impact on the delivery date, it must be identified and stated explicitly. Dependencies could be of four types:

- **Start-Start :**

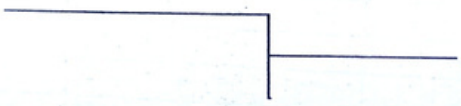


Activity 1 and Activity 2 must start together. A good example for this type of activity in the software life cycle in program specifications and Unit Test Plans - it is best if both these activities are started together in parallel. This kind of relationship can be diagrammatically represented as shown below.

- **Start-Close :**

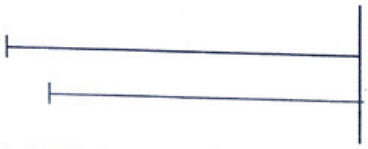
In a Start-close relationship, Activity 1 must start before Activity 2 is closed. Typical example of this activity in a software life cycle User Documentation must start before document is completed. This type of relationship is diagrammatically represented as shown below.

- **Close-Start :**



Activity 2 can start only after Activity 1 is closed. This is the most critical dependency as it has the maximum impact on end delivery date. An example of such a relationship is that User Acceptance Test can only start after System Test is completed. This type of relationship represented diagrammatically as shown below.

Close-Close :



In this type of relationship, the two activities must end simultaneously or together. A typical example of this type of relationship is the completion of User documentation and Acceptance Test. The User documentation is reviewed during Acceptance test and thus both these activities have to end together.

It is important to identify the key relationships which impact the end delivery date and not every minute dependency just to make the planning complicated.

Based on the effort estimation done for the project, the effort must be apportioned to each and every activity. The effort for each activity will depend on the complexity of the work, the knowledge level and familiarity of the people working on the activities and uncertainties (like unclear specifications) associated with the activity. It must be noted that the sum total of the effort for all the activities must be equal to the total estimate done for the project. If there is a deviation of more than say 10 percent, there must be a review of the apportionment of the estimate to each activity. If it is found to be OK, then approval of estimates for the revised estimate must be obtained from the sponsors of the project.

The Project Schedule chart which has been built so far will appear as shown below. It must be noted that we are using the summarized version of table for further discussions keeping simplicity in mind.

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
1. Project Initiation					
1.1 User Request		1			
1.2 User Request Review		1			
1.3 Feasibility Study		5			
1.4 Estimation		3			

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
1.5 Approval of Estimates		3			
1.6 Risk Analysis		2			
2. Planning and Tracking					
2.1 Initial Plan	1.5, Close-Start	5			
2.2 Tracking & Review		20			
2.3 Revision of Plan		10			
3. Life Cycle Activities					
3.1 Functional Specifications		50			
3.2 Design Specifications		50			
3.3 Development Close-Start	3.1,3.2				
3.3.1 Program Specifications		40			
3.3.2 Unit Test Plans		40			
3.3.3 Coding		60			
3.3.4 Unit Testing		60			
3.4 System Test Plan		30			
3.5 User Acceptance Plan		30			
3.6 Documentation Close-Close	3.8, 30				
3.7 System Testing	3.4, Close-Start	45			
3.8 User Acceptance Testing	3.5,3.7 Close-Start	30			

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
3.9 Packaging and Release		10			
4. Implementation					
4.1 Planning		5			
4.2 Testing Conversion		30			
4.3 Freezing data		3			
4.4 Database Definition		5			
4.5 Transfer to Production	3.9, Close-Start	3			
4.6 Conversion		10			
4.7 Parallel Run		20			
4.8 Live Operations		1			
4.9 Post Implement Support		90			

4.5.2. Allocation of Responsibilities :

After identifying all the activities for the project, apportioning the effort and locating dependencies, the next step in Project Scheduling is allocation of various activities to people. This is very important task and if it is not done properly, it could break an otherwise successful project. People are the cornerstone or the most vital assets of a project team. If they are given responsibilities which are not in line with their skills, we could easily end up with a square peg in a round hole. It s not always possible to find a perfect resource for each and every activity. It is however imperative that the best of available resources is put for the most appropriate task. If somebody has good testing skills and application knowledge, the person must be put on writing Functional Specifications and Test Plans and not on design and program specifications. It is also possible that there may be a gap between skills required for an activity and possessed by an individual. In such a case, it must be judged if the person can pick up the required skills quickly. An accelerated training program for the skill gap is a must for the success of the project. If a person is managing a team for the first time, the person may not be aware of Project Management requirements and activities. In such a case, the person must be trained, close supervision in the early stages will help the person adjust to the new role quickly. People are generally reluctant to admit that they do not possess some skills. It is for the management to understand these deficiencies and take the required

corrective action. The work experience of an individual and appraisal records should give enough information on the skill level of an individual. The problem faced by most organization is that the do not have an organized skills inventory and thus selection of the right individual for the job becomes more of potluck than a structured exercise. Apart from skills inventory, the organization must also have the details of the current assignment and the time when the person will be available for the next assignment. A resource calendar is thus required for allocating people to various activities.

Another key aspect of skill building is timing. There is no point in building skills after the event. If a person is going to do programming in C for the first time, the person must be trained on C language and it's associated programming standards before writing the first program. If it is done after the person has written a few programs, then the person may have to 'unlearn' and 'relearn' which is much more difficult than just learning. This is a very common mistake made by most people - putting people on the jobs without the proper training and then realizing the mistake half way through. This is because most people think that the person will pick up skills on the job. In reality, once the person is on-the-job, the regular day-to-day activities drown an individual and the person keeps doing rather than learning.

We have discussed so far the resource allocation criteria and training requirements. While these are critical from the overall project deliverables, the one aspect where the senior management must put a lot of thought is the selection of the Project Manager. 1.1 the Project Manager is selected through objective selection criteria, then the person will make up for the weakness in many other individuals of the team. If the Project Manager possesses all the skills, then the person will automatically realize the importance of planning, training, reviews etc. and implement them in a timely manner.

Another factor which must be considered during resource allocation is Resource leveling. In many organizations, people work across many projects or have some corporate responsibilities. In such cases, consideration must be given to see that there is no overloading of a resource. If resources are overloaded on a continuous basis, it results in breakdown of an individual - this results in poor quality of outputs or poor productivity and ultimately in poor morale. It also leads to a high degree of attrition in an organization. If a particular person is getting overloaded, then some work load must be reassigned to another person. This is known as resource leveling whereby we ensure that all resources are used to the optimum level and not overloaded.

Software organizations have a high degree of volatility in their manpower. Software people are a very mobile community. It is therefore important that backup resources are provided for critical positions in the project team. It is not feasible and practical to provide a backup resource for every person in the project team. However, critical persons like Project Manager, Database Administrator etc. must have a backup so that work does not get disrupted in case of a change in person. For other

People in the team, organizations normally maintain a software pool for replenishment in case of major attrition in a project.

The final action in Resource Allocation must be to communicate the Roles and Responsibilities of an individual in the project lack of communication has been identified as one of the most common causes of failure. Roles, Responsibilities, Goals and Appraisal criteria must be communicated clearly at the time of assignment of a person to a project. If this is not done, it leads to the common Everybody-Somebody-Nobody syndrome-Everybody thinks somebody will do the job. Eventually nobody ends up doing it. The Project Schedule table given above must be circulated to one and all so that everybody knows who is responsible for what in the project.

After allocation of responsibilities, the table will look as shown below.

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
1. Project Initiation					
1.1 User Request		1			
1.2 User Request Review		1			
1.3 Feasibility Study		5			
1.4 Estimation		3			
1.5 Approval of Estimates		3			
1.6 Risk Analysis	1.5	2	PM		
2. Planning and Tracking					
2.1 Initial Plan Close-Start	1.5,	5			
2.2 Tracking and Review		20	PM, TL Users		
2.3 Revision of Plan		10	PM		
3. Life Cycle Activities					
3.1 Functional Specifications		50	PM, TL, Users		

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
3.2 Design Specifications		50	PM, TL Reviwers		
3.3 Development Close-Start	3.1,3.2				
3.3.1 Program Specifications		40	Developers		
3.3.2 Unit Test Plans		40	Developers		
3.3.3 Coding		60	Developers		
3.3.4 Unit Testing		60	Developers		
3.4 System Test Plan		30	PM, TL		
3.5 User Acceptance Plan		30	Users		
3.6 Documentation Close-Close	3.8,	30	PM Technical Writer		
3.7 System Testing	3.4, Close-Start	45	PM, TL, Developers		
3.8 User Acceptance Testing	3.5,3.7 Close-Start	30	User, PM TL, Developers		
3.9 Packaging and Release		10	PM,TL		
4. Implementation					
4.1 Planning		5	PM, Users		
4.2 Testing Conversion		30	PM, Users		
4.3 Freezing data		3	Users, PM		
4.4 Database Definition		5	PM, DBA		
4.5 Transfer to Production		3	PM, Librarian		
4.6 Conversion		10	PM, Users, TL		

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
4.7 Parallel Run		20 TL	PM, Users,		
4.8 Live Operations		1	PM, Users		
4.9 Post Implement Support		90	PM, Support Team		

In the table above, the responsibilities have been assigned to generic classes of resources. In real life, the names of individuals doing the particular task would be mentioned. It is possible that activities can be reshuffled across people during the life of the project based on the progress of the project. In case the planning was being done at a more detailed level, then the author, reviewer, coder and tester of each program would have to be stated explicitly in the plan.

The generic abbreviations or class of people used above are described below:

- PM - Project Manager
- TL - Team Leader responsible for a particular module or activity within the project
- Users - The people who are the ultimate beneficiaries of the system. They define the requirements of the system'
- Developers - People responsible for writing program specifications, coding UTPs and Unit testing
- Technical Writers - Specialist documentation people who translate the system functionality into a User manual
- Support Team - the team responsible for supporting the system after it has gone live
- DBA - Database Administrator
- Librarian - Person responsible for maintaining the source code library

It is also the responsibility of the Project Manager to develop an Organization structure for the project team. This will enable people in the project team to know whom they should approach in case they have difficulties in doing activities assigned to them. The difficulties could be relating to lack of resources, know-how or administrative in nature. Refer to Hierarchy Chart discussed under Role of System Analyst.

4.5.3. Scheduling of Activities :

The next stop in Project Scheduling is to schedule the various activities so that we can arrive at the final delivery date of the project. Scheduling of Activities must take into consideration the following aspects:

- Dependencies between activities
- Availability of Resources
- Holidays and Vacations
- Contingencies
- Rework and Iterations
- User Requirements

Let us look at some of the points to be considered for each one of the above.

Dependencies Between Activities : We have already identified the dependencies between activities when we did Identification of Activities. When we schedule activities we must take cognizance of the dependencies. Most Project Management tools provide for automatic linkage of dependencies and the Project Manager is forced to schedule activities based on the defined dependencies. The dependencies are best understood through diagrams rather than tables or textual descriptions. Common diagramming techniques like the GANTT chart or Network diagram are described under 'Pictorial Descriptions'

Scheduling of a software project does not differ greatly from scheduling of any multitask development effort. Therefore generalized scheduling tools and techniques can be applied to software with little or no modification. The Program Evaluation and Review Technique (PERT) and Critical Path Method (CPM) are two project scheduling methods that can be applied to software development. Both techniques develop a task network description of the project. It involves a pictorial description of the activities from beginning to end of project. The network is defined by developing a list of activities (or tasks), sometimes also called the project Work Breakdown Structure (WBS), which are associated with a set of dependencies or orderings that indicate the sequence in which the tasks must be executed. Both, PERT and CPM provide quantitative tools that allow the software planner to:

- Determine the critical path - the chain of path that determines the total elapsed time of the project
- Establish most likely time estimates for individual tasks by applying statistical techniques
- Calculate boundary times that define a time window for a particular task.

Availability of Resources : Another aspect of project scheduling relates to availability of resources to execute the project. In this section, we are only dealing with human resources required for the project. Other resources required for the project are discussed under Resource Planning. It is possible that the people who are allocated to the project are not available from the start date as planned by you. They may be busy in some other critical activity and cannot be spared for the project. Under such circumstance, either the start date must be delayed till the person is made available or alternate resources must be mobilized if schedules cannot be pushed ahead. Resource mobilization at short notice can only happen through subcontracting the work to contract programmers. If permanent resources are to be

recruited then the lead time for recruitment must be factored into the scheduling exercise.

Sometimes resources are available on a part-time basis. If this is the case then, the effective manpower availability is only half or Quarter depending the amount of time the person will be spending on the project. The equation, which must be borne in mind, is given below:

Effort = Number of working days between
Plan-Start-Date and Plan-End-Date
Multiplied by
Effective number of available resources

The key points to be noted in the above equation are the following:

- The number of days is the number of working days and not calendar days
- The number of resources is the effective number of resources and not the physical number of resources. The effectiveness may be reduced on account of part-time availability or the skill level of an individual.

In project scheduling, the equation is used to calculate the number of working days required for completing the task, as the effort and number of effective resources are known. Once the numbers of working days are computed, the end date of the task can be calculated once the start date is fixed. It is normal practice to start an activity as early as possible after dependency conditions are satisfied. However, this may not always be feasible, as resources may not be available. Thus, the end date is always calculated based on above factors.

Holidays and Vacations : While planning, it is a good practice to take into account all holidays and planned vacations that are going to be taken by people who are allocated to the project. If the above details are not fully available at the planning stage a certain percentage must be provided as buffer for these activities. This is particularly the case when you are dealing with international project teams where work on different modules are done in different countries and full details on holidays and vacations may not be available at the planning stage.

Contingencies : It is nice if everything works out perfectly. But experiences on software projects have shown that things always don't go to a plan. You must provide for some unforeseen contingencies attrition of people, illness or personal emergencies, fire fighting in some other areas within the organization etc. If the contingencies do not occur, then the project will complete ahead of schedule, which is good from all aspects. The idea is to present a realistic picture rather than an optimistic picture of the completion of the project.

Rework and Iterations : An area, which is most often ignored by Project Managers, relates to rework or iterations due to some errors in design or specifications. We are not talking about changes to specifications, which will need revision of plan in any case. We are talking about incorporating review comments

and errors found during testing. It is a common practice to have two rounds of system test, as the first round is normally a very turbulent one. Similarly, FS, Design, Test Plans and Documentation have reviews and time must be provided to ensure that all review comments are tracked to completion.

User Requirements : We do not live in an idealistic world. If after doing all the above scientific scheduling, we arrive at an end, which does not serve its purpose. If RBI has sent a circular that all banks must submit a foreign exchange transaction report in a particular format with effect from the first of January and based on your scheduling exercise, the system is going to be ready only in mid February, the system is not meeting the business needs. In such cases, an effort must be made to see if the elapsed time of some activities can be crashed. Crashing can be done by one of the following:

- Arranging for extra resources to meet peak time loads
- Arranging for shift operations to speed up end delivery
- Use of productivity tools like automated testing tools, coverage analyzer, debuggers, etc.
- Adding more processing power to machines

The techniques described above may result in some reduction of elapsed time but one must remember that it is not possible to pour five liters of milk into a one-liter can. There is a limit beyond which elapsed time cannot be reduced. It must also be noted that when elapsed times are reduced, the risks associated with the project, specially relating to timely delivery are higher.

At the end of the entire Project scheduling the table will appear as shown below.

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
1. Project Initiation					
1.1 User Request		1		25-6-1996	25-6-1996
1:2 User Request Review		1		26-6-1996	26-6-1996
1.3 Feasibility Study		5		01-7-1996	10-7-1996
1.4 Estimation		3		15-7 -1996	20-7-1996
1.5 Approval of Estimates		3		21-7-1996	31-7 -1996
1.6 Risk Analysis		2		21-7 -1996	31-7 -1996
2. Planning and Tracking					
2.1 Initial Plan	1.5, Close-Start	5	PM	1-8-1996	1 0-8-1996
2.2 Tracking and Review		20	PM, TL, Users	1-8-1996	31-12-1996

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
2.3 Revision of Plan		10	PM	1-8-1996	31-12-1996
3. Life Cycle Activities					
3.1 Functional Specifications		50	PM, TL, Users	1-8-1996	31-8-1996
3.2 Design Specifications		50	PM, TL, Reviewers	1-9-1996	30-9-1996
3.3 Development	3.1,3.2 Close-Start				
3.3.1 Program Specifications		40	Developers	1-10-1996	15-11-1996
3.3.2 Unit Test Plans		40	Developers	1-10-1996	15-11-1996
3.3.3 Coding		60	Developers	15-10-1996	30-11-1996
3.3.4 Unit testing		60	Developers	15-10-1996	30-11-1996
3.4 System Test Plan		30	PM, TL	10-10-1996	15-11-1996
3.5 User Acceptance Plan		30	Users	15-10-1996	20-11-1996
3.6 Documentation	3.8, Close-Close	30	PM, Technical Writers	1-11-1996	31-12-1996
3.7 System Testing	3.4, Close-Start	45	PM, TL, Developers	1-12-1996	20-12-1996
3.8 User Acceptance Testing	3.5,3.7 Close-Start	30	Users, PM, TL, Developers	21-12-1996	31-12-1996
3.9 Packaging and Release		10	PM,TL	1-1-1997	10-1-1997
4. Implementation					
4.1 Planning		5	PM, Users	1-12-1996	15-12-1996
4.2 Testing Conversion		30	PM, Users	1-12-1996	31-12-1996
4.3 Freezing data		3	Users, PM	15-12-1996	31-12-1996
4.4 Database Definition		5	PM,DBA	1-1-1997	10-1-1997
4.5 Transfer to Production		3	PM, Librarian	11-1-1997	12-1-1997
4.6 Conversion		10 TL	PM, Users, 13-1-1997	13-1-1997	
4.7 Parallel Run		20 TL	PM, Users, 14-1-1997	21-1-1997	

Activities	Depend-encies	Effort	Resources	Plant Start Date	Plan End Date
4.8 Live Operations		1	PM, Users	22-1-1997	22-1-1997
4.9 Post Implement Support		90	PM, Support Team	22-1-1997	31-3-1997

4.5.4. Identifying Milestones :

There must be reviews at various stages of the project to ensure that the system, which is being developed, will meet user requirements, will be cost effective and will be delivered in acceptable time frames. Periodic reviews say fortnightly or monthly are a must. The periodicity may have to be increased as the final date gets closer or if the project has run into troubled waters. The objective of these reviews must be to identify bottlenecks or issues related to the project and take corrective action so that the project can be brought back on track or completed with minimum delay or cost overrun.

In addition to periodic reviews, it is important that phase reviews or milestone reviews are also conducted. These reviews are conducted at the completion of a milestone. Thus milestones must be identified in the system life cycle. Typical milestone reviews are review on completion of Functional or Design Specifications, on completion of Development, after User Acceptance Testing etc. The Project Schedule must indicate those activities, which are milestone activities. Reviews at the end of milestones must involve users as it involves movement of the project from one phase to another. Corrective actions will become costlier as the project gets into move advanced phrases and hence it is important that milestone reviews are conducted punctually and with all parties present.

4.5.5. Pictorial Descriptions :

There are two common forms of pictorial descriptions used in Project Management. They are:

- GANTT Chart
- PERT/CPM Network Diagram Let us look at them one by one.

GANTT Charts : These charts are plotted with time as X-axis and bars to indicate the planned completion dates and actual progress for all the activities in the project. Most Project Management tools offer facilities for automatic generation of GANTT charts from the data supplied in tabular format. These are very difficult to draw manually for complex projects where numbers of activities are very large. In order to effectively use the GANTT chart, it must be noted that it must be generated at a summary level so that an overall picture of the status of the project is known. If the GANTT chart is so complex that the printout covers the entire wall of the room, then it will be extremely complex to analyze. A simple GANTT chart at summary level will help in identifying critical bottlenecks and management can focus their attention on those aspects. A typical GANTT chart for the activities in the life

cycle stage of the project from the project from the project schedule is given below.

Activities	Aug 96	Sep 96	Oct 96	Nov 96	Dec 96	Jan 97
Functional Specifications	1/8	31/8 11/8	11/9			
Design Specifications		1/9 10/9	30/9 19/10			
Development Program specifications Unit Test Plans			1/10 20/10 1/10 20/10		15/11 5/12 15/11 5/12	
Coding			15/10	30/11		
Unit Testing				15/10	30/11	
System Test Plan			10/10	15/11		
User Acceptance Plan			15/10	20/11		
Documentation				1/11	31/12	
System Testing					1/12 20/12	
User Acceptance Testing					21/12 31/12	
Packaging and Release					1/1	10/1

In the above GANTT chart planned schedule is given in gray while actual progress is given in white. The dates given on the GANTT chart are the start and end date of each activity. If you have a sophisticated Project Management Tool, the resources allocated to each activity, the planned effort and actual effort can also be printed alongside. As you can see from the above chart, it gives you a bird's eye view of the status of the project. The management can then focus on the problem areas and find solutions to overcome the problems after discussion with the Project Manager.

PERT/CPM Network Diagram : This is a diagram, in which all activities are shown as a network of dependent activities. The effort for each activity is estimated using a Pessimistic, Realistic and Optimistic approach. The most likely date for completion of the activity is calculated as

$$\frac{P + 4R + 0 - 6}{-6}$$

The arrows in the network represent activities while the nodes represent completion of activities or milestones. If two arrows end on the same node, it implies that both activities must end before the activity emerging from the node can start. A typical but simplified PERT/CPM network diagram is shown below:

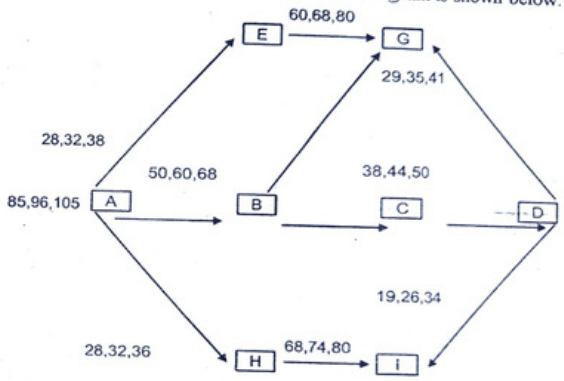


Fig. : 1

In the diagram given above, each activity is represented by an arrow and each node is a milestone. The figures are the optimistic, realistic and pessimistic estimates for each activity. For each activity, you have an Earliest Start Date and Latest Start Date. Earliest Start Date is the earliest date on which the activity can start based on the dependencies of the various activities. Latest Start Date is the date on which the activity must be started so that it will be completed on time and will not affect the overall delivery schedule of the project. It is a good practice to start on the Earliest Start Date but it is always not practical to start on the Earliest Start Date due to a number of constraints. If the activity starts later than the Latest Start Date, then it will impact the overall delivery of the project.

The period between the Earliest Start Date and Latest Start Date is called the Float. If the activity is started within the float period, it will not impact the overall delivery of the project. The rationale for starting on the Earliest Start Date, if feasible is to provide for incorrect estimation of effort, contingencies etc.

The most important item in the PERT/CPM Network diagram is the Critical Path, which is explained in the next section.

4.5.6. The Critical Path :

You will notice that the diagram above has two kinds of arrows - black and gray. The gray arrows are activities with float. Black Activities are activities without float. In activities which do not have a Float, the Earliest Start Date and Latest Start Date are the same. There is no float or 'free' time available in these activities - they have to start on the specified day. Any delay in starting an activity with zero float will result in delay of the final delivery of the project. The path of activities with zero float from the start of the project to the end of the project is called Critical Path. This path is shown in black in the diagram. Activities-on the critical path need to be monitored with more attention as any delay in these activities will delay the overall completion of the project. If an activity on the critical path gets delayed and the project has still got to be completed on time, then the future activities have to be crashed. In order to still meet the deadline, the activities, which need to be crashed, are the activities on the Critical Path. Crashing involves use of non-conventional and extreme measures like getting resources on contact, working overtime or on holidays, resorting to shift system etc. To ensure timely completion. If crashing is done on a project, it will definitely escalate the cost of the project.

5. Resource Plan :

You have already seen the various aspects relating planning of Human Resources. In this section, we will focus attention on other resources required for the project. Typically, a software project will need a combination of the following items:

- Hardware
  - Servers
  - Workstations
- Software
  - Environment Software
  - Third Party Software
  - Included Products
- Tools
  - Version Control Tool
  - Testing Tool
  - Standards Tool
  - Code generators
- Communication
  - Data Communication Equipment
  - Data Communication Software

The list given above is not comprehensive but a typical one for a software project. It is the Project Manager's responsibility to ensure that the resources required for the project are made available in a timely manner as people without supporting resources could lead to idle manpower and thus wastage of human resource utilization. Points that must be taken into consideration for the above resources are given below :

5.1. Hardware :

Hardware required could be the backend machine (server) and the front end workstations (Personal Computers or dumb terminals). The number of machines and their configuration are important. Insufficient number of machines could lead to loss of productivity. If the server machine is a look-alike of the final target machine and not an exact replica, then time must be provided for last minute patches which may be required to get it working on the target machine. If the configuration of the machines is lower than the required specifications, some of the software may not work properly and some may take longer time to develop or execute, thus resulting in loss of productivity. In case the server is being used for other projects also, the disk space requirements on the server must be specified - otherwise development could slow down to non-availability of disk space.

5.2. Software :

You need Environment software like operating systems, database and front end tools for development of software. The correct version of these tools must be available for development. If the software is developed on a different version, then effort will have to be put in to port the software to the new version before the software is moved to production. Adequate time has to be provided for this in the Project Plan. However, a bigger weakness under such a scenario will be that the software would not have used all the features of the target version and could thus have some design inefficiencies.

It is also important to have the required number of licensed copies for development and production environment. For development, you will need development version of the environment software. In Production, only the run time or executables may be required. These must be planned and procured in time. Training on the environment software should also be provided to those people who are going to be working on the development and do not have adequate knowledge of the environment software.

You may also have some third party software which interfaces with your product. Let us assume that you are developing a Banking software. Your customer wants to use a signature verification package from a different vendor, as the users are very familiar with it. In this case, you will need a copy of the third party software during development so that the interface with the third party can be properly tested. The procedures for getting a copy of the third party software for development and support must be organized. If the third party software is being upgraded when your project is under development, then it must be ensured that the system is tested with the final version of the third party product before delivery of the system.

Just like third party software, you may also need to include some software in your system. Continuing with the banking system example, the bank may have some proprietary code for encrypting and storing passwords. The bank will only give you the executable version of the software. This software will have to be included in your final product as password validation may have to be done from various modules in your system.

Tools :

Tools will be required for a number of functions within the project. You may a tool for version control of software, for project management, for testing and document management. Normally, organizations have standards for usage of tools. In such cases, the organizational standard must be followed. However, in me cases, the user may insist on a tool used in their environment. Then tools will to be either procured or got on loan from the client. In either case, it must be sured that people using tools must be provided adequate training and support on f tools before they actually use it on the project.

t. Communication :

In case the project involves use or development of data communication software local or wide area networks, the required equipment like modems, multiplexers, ptors must be procured in a timely manner. In some cases, if the actual ipment cannot be procured due to some practical constraints, then simulation niques will have to be done for testing of software. In addition to data mmunication equipment, software like terminal emulators, file transfer protocols, tail etc. May be required for interfacing and testing with the system, which is ing developed. The Project Manager must ensure that these equipments are ilable in a timely manner for the project.

roject Tracking and Oversight :

There's an old saying that' Drops of water make an Ocean'. A one day slip in ct schedule will rarely be fatal to a project. It is when the days and add up, and er the length of the project, that small delays can result in big problems. In order oid the big problems. In order to avoid the big problems, Project Tracking and rsight on an ongoing basis is essential. Planning without Tracking is like planning d in concrete - you will not get any fruits for your labour. The essential items proper tracking are the following:

- Schedule Tracking
- Resource Tracking
- Cost Tracking
- Management Oversight

t. Schedule Tracking :

Tracking of schedule of projects is normally a Bottoms-up process. Unlike nning, which is a Top Down process, Tracking is normally the reverse. In tracking, ata on actual progress is gathered at the lowest level of activities. Each grammer must input the effort spent on allocated activities along with the status ch activity. This information is then consolidated across various activities to pare reports for management reporting and review by Project Manager for tive actions. The most common method used for gathering actual progress inst various activities is Time sheets. Time Sheets are successful only when are automated and linked to the plan so that automatic comparison is available ween Plan and Actual progress. Typically, Time Sheets capture the following mation:

- Actual effort spent on the activity
- Status of the Activity
  - Closed (Activity is fully complete)
  - Percentage Completed
  - Days required for completion

Reasons for Delay : Most software professionals do not like to enter timesheets as they feel that the information will be used against them in their performance appraisals in case some activities have gone beyond estimated time frames. Timesheets have been successful only in organizations which have a culture that this information will be used only for Project Tracking and nothing else. If this culture is prevalent, then you will get accurate information in the time sheets. Otherwise people tend to manipulate information on timesheets to protect their personal interest.

Another aspect of timesheets is that it must be entered within acceptable timeframes. If people don't enter timesheets at defined periodicity, then the Project Manager and Management will not know the true status of the project as the reports will reflect an old status of the project. Enforcement of timely entry of time sheets has to again be a cultural change in the organization and must be enforced as a discipline. The Project manager must have access to information of people who have not entered time sheets so that he can follow up with them and enforce the required discipline.

Once all the people have entered the time sheets, the Project manager can consolidate the information and produce summary reports for circulation and Analysis. This information must be analyzed as described under Management Oversight. It is not sufficient if the information is circulated and filed as information to all concerned.

6.2. Resource Tracking :

Utilization of Manpower resources is tracked using Time sheets as described above. However, tracking of utilization of other resources has to be a combination of manual and automated processes as per norms within the organization. Typically, the following information needs to be tracked:

- Number of machines used by the Project
- Number of Software licenses utilized by the project
- Disk Space utilized by the project

The actual utilization information can be gathered by automated or manual means. In order to know the actual project cost, it is important that the resource utilization is tracked by some logical means. It is not required to split hair over minor deviations in resource utilization as these are very difficult to measure. What is required is an overall process which has at least 80 percent accuracy in information.

5.3. Cost Tracking :

The actual utilization of Manpower and other resources must be translated to cost at the same unit cost that was taken during Project Planning when cost estimates were made. We can have another column which is based on actual costs which is apportioned over the resources utilized by the project. This may be lower or higher than the unit cost which was estimated at planning stage. Thus the following costs must be computed for resource utilization:

- Cost at Unit cost used for planning
- Cost based on actual expenses incurred for the resources

The first cost can be compared with the cost estimates made at the planning stage to see if the project is still within approved costs. In order to arrive at the total project cost, other costs which are not covered above, must be added. Typically, this will include items like Travel, Project lunches, local conveyance etc. If the total cost of the project exceeds the total project cost on a pro-rata basis, then one must review it to see whether we need to take re-approval of estimates from project sponsors.

5.4. Management Oversight :

Management must review software projects at periodic intervals to ensure that corrective measures from management side are taken to resolve project management related issues. Management Reviews of Projects are normally held at monthly or fortnightly intervals depending on the policies of the organization. It might also be worthwhile to have a review on accomplishment of a milestone in the project. These meetings must be attended by senior management personnel, and heads of Infrastructure, Quality Assurance, Finance, Marketing with a two fold objective - they are aware of the project status and secondly, they can jointly resolve issues so that project schedules are not impacted. The points, which are normally discussed in such meetings, are the following:

- Some activities in the Project are behind schedule. What is the reason? How can it be corrected? Will it have an impact on the final delivery date? Does customer have to be informed of the delay?
- Is the Project still under budget or have costs gone beyond estimates. If costs have gone beyond estimates, what will be the final cost? Who will bear the overrun? How can it be minimized?
- What do the Quality metrics of the Project indicate? Is testing being done in organized and systematic manner? Are test plans being written before testing or are they just being documented for compliance?
- Are there too many changes to Requirements? Is it being controlled? Is customer being told the cost and schedule impact of these changes?
- Does the project have a resource or manpower problem? If yes. What is it? What should be the corrective action?

At the end of the meeting, a detailed action plan should be drawn out indicating the corrective measures required to bring the project back on track. If the project has already lost so much ground that it will definitely have an impact on the end delivery date, then the user or client must be informed of the delay.

7. Project Metrics :

We live in the Era of Excellence. In the world of quality, an oft quoted slogan is "Can't Measure; Can't Improve". In other words, it means that you can only improve in areas where quantitative measurements are possible. Subjective data always leads to incorrect inferences. Similarly if the objective data is suspect, then you will end up taking incorrect decisions. The quantitative data must follow the 80-20 rule - it should be about 80 percent accurate. The cost for improving the accuracy of measurement for the last 20 percent may be higher than the cost of the project itself and may not thus be cost-effective in taking decisions related to the Project. The key metrics to be measured are the following:

- Schedule & Effort Metrics
- Quality Metrics
- Cost Metrics

In all the above metrics, you need to take the absolute value of the data of the project and compare it with the plan and if the organization is maintaining a project history database, then it should compare the data with historical metrics to arrive at realistic conclusions based on historical data. Let us look at each one of the above metrics and see how they can be used for monitoring the project.

7.1. Schedule & Effort Metrics :

The key data for these metrics are a comparison of the following:

- Planned effort and Actual Effort
- Planned Start-End date and Actual Start-End Date

The first comparison will give us the Effort Variance while the second comparison will give us the Schedule Variance. Let us say that the Actual Effort has exceeded the Planned Effort - in this case it is a negative variance as the cost of the project is going to be higher than estimated. In such a situation, it might be a good idea to look into the historical database of similar projects and compute an effort Predictability factor as follows:

Actual effort less Planned Effort
divided by
Planned Effort

This must be computed as a percentage and averaged out across all similar projects. This will give you a trend and perspective for the project. If it works out that the above average deviation is negative by 28 percent and the current project is running at a negative deviation of 15 percent, then you can expect that this project will have a further deviation as it progresses. This does not imply that one should stop trying to bring down the effort deviation - it is only an indicator of likely events

to follow. The long term corrective measure is to review the estimation process, the planning process and productivity metrics.

The second comparison will give us the slippage in terms of elapsed days. If the actual end date is likely to be after the planned completion date, then we? Know that there is probably going to be a delay in the completion of that activity. If that activity happens to be on the critical path, then it will have an impact on the final delivery date as well. Again, we can do a comparison!", with historical data as stated in Effort Predictability.

Another metrics which would be on interest to Management relates to Productivity. Normally, Productivity is measured as Number of tested lines of code produced per day or number of function points produced per day. This figure must be measured and reported to Management for every review meeting, as it will give indication if there are people related or Changes related issues in the project. If this figure deviates significantly, positively or negatively, from the historical data, there is cause for concern. If the productivity is much higher, then process adherence and testing process or understanding of requirements must be investigated. Alternatively, if there are some logical explanations like the use of code generators or testing tools, then leap in productivity is understandable. If the productivity figure is very low, then it could be on account of lower skill level in the personnel working on the project, frequent changes in requirements being incorporated in an ad-hoc manner or some inter-personnel issues in the project team.

7.2. Quality Metrics :

Quality Metrics refers to measurement of the Quality of the system. Typically, this is measured in number of errors per 1000 lines of code or function points. The most important aspect in this metric is the definition of various terms and consistent application of the definition. Some of the key issues are as follows:

- How do you define an error in a system? It is possible to combine multiple errors and report them as one error. There must be an objective definition of an error. There must thus be an objective definition of an error. There are no proven rules for this definition but each organization must define it objectively and more importantly, use it consistently.
- How do you define lines of code? Do you include comment lines and other non-executable lines of code. What if some programmers have written multiple statements in one line while another programmer has written one statement per line. Again, clear organization definitions are required.
- How do you define lines of code when you are making a modification to the system? Is it just the number of lines which were changed? Or is it the total number of lines across all programs which were modified.

There are no easy answers to the above questions. The industry is itself evolving various ground rules for the above. The key is consistent application of organization definitions without frequent changes of definitions.

If we compare this figure with historical data, we will get some idea about the level of testing and Quality of code. In this case also, if the deviation is outside an acceptable band, then there is a need for introspection. If the number of errors in testing is very low, then the test plans are probably not adequate and coverage of testing is poor. In all probability, the same code is being tested again and again and large chunks of code or skill level of programmers comes into question. If it is within the acceptable band, then the testing is proceeding properly and no corrective action need be taken.

The historical data used for comparison should be the average of a number of projects in similar environment.

7.3. Cost Metrics :

This is really the role of the Finance manager. The inputs for calculating costs have to be provided by the Project Manager. Based on the inputs given by the Project Manager, the Finance Manager calculates the actual cost of the project based on unit costs supplied at planning stage and actual expenses incurred. These expenses are compared with the budgets allocated to the project to see if the project is under budget or over-budget. If it is outside an acceptable range, then corrective actions need to be taken. If the project cost is much lower than the budget cost, then the estimation process must be reviewed. If the project cost is very high, estimation process must be reviewed along with tighter monitoring of cost to increase the cost effectiveness of the project.

8. Project Closure :

Projects can be closed on completion of all activities or prematurely if it is abandoned due to a change in business priorities or some other reason. In either case, the project must be closed properly so that all its data is logged into the historical database. If this is not done, the historical data will not get updated and will not reflect the true picture. Irrespective of whether it was a normal or premature closure, the following activities must be completed:

- A letter of project completion must be obtained from the customer
- Details regarding Schedule, Effort, Productivity, Quality and Cost Metrics must be entered into the system.
- All information regarding the project must be circulated to impacted parties.
- All documentation must be put into the library or Document management system
- All software configuration items must be logged into the Source Code library
- A Release note indicating the release of all resources must be sent to the appropriate managers.
- All items which are not required must be deleted from disk
- Team members must be communicated of their next assignment

Before disbanding the project team, a formal meeting must be held where team members must be given an opportunity to share their experiences on the project.

s can give valuable inputs to improve the performance on the next project. This st be done in addition to formal performance appraisal which is done on a one- one basis.

Miscellaneous Items :

You can read the best book on Project Management, attend the best training gram on Project Management but there is no substitute for practical experience. have to be in the hot seat to experience the ecstasy and disillusionment, the piness and the sorrow, the relief and the anxiety and a number of such feelings. e day the Project Manager may feel that he is on top of the world and everything ailing smoothly but one minor incident like the user asking for some additional ds or some key people leaving the project or a malfunction in some hardware ources can just leave you feeling at the other end of the tube at the end of the day. e best look alike of a Project Manager from the non-software world is the juggler. has to keep juggling his resources and be on the move all the time to ensure ximum resource utilization and clearance of all bottlenecks. In this section, we k at some of the juggling acts that a Project Manager may be called upon to do ring his stint as Project Manager. The topics discussed are:

- People Management
- Resource Management
- Quality v/s Schedule
- System Integration Issues
- Idealism v/s Realism
- Sub-Contractor Management

Let us just briefly look into each one of the above.

1. People Management :

Software Projects are successful if the Project Manager is an excellent People anager. Executing software projects is all about getting the best out of people as team. There may be a few technically brilliant people, there may be a few who ow the application inside out, there may be a few whose dedication and mmitment are unquestionable, and so on... but the key is to harness this group of verse people to think along similar lines and work towards a common objective. ll inter personal problems and conflicts must be resolved as soon as they come to e knowledge of the Project Manager. Doing timely appraisals and sending people training at the time when it is required brings that additional commitment from ople. Periodic socializing in the form of project lunches and dinners, picnics or Fsite meetings make people feel that they are important. A pat on the back when meone has done a reasonable job make one feel that his work is noticed. These e easy to do when the going is good. The real test of a project manager is when is against some odds.... if the Project Manager can maintain his cool and get the st out of his team even under trying circumstances, the team is a sure shot winner the-way. The way to success in a software project is Teamwork.

9.2. Resource Management :

Probably next in criticality to people management is Resource Management. Software projects are always executed under very tight resource constraints. Workstations are in short supply, disk space is a constant irritant, not enough licenses of the compiler... these are standard complaints you are likely to hear in a software organization. There are many reasons ranging from budgetary constraints to system integration constraints which could be rationale for the situation. The role of the Project Manager is to understand the constraints and try to find solutions within the constraints rather than stating, 'My project is in trouble. I asked for X and I am not even getting .5 X'. Some of the common ways of solving resource management issues relates to restoring to shift operations so that resources can be shared and there is better utilization. Operating in shifts has a big negative whereby everybody does not get to know each other as they will be working at different times. Though the Project manager must accept and work within the given constraints, it is the Project manager's responsibility to keep Management aware of the ideal requirements and request for them as soon as possible. The thrill of completing a project successfully under heavy resource constraints is unparalleled.

9.3. Quality v/s Schedule :

There is an oft-quoted saying 'I want it Good, Fast and Cheap'. Most marketing people say that there is a contradiction in the requirement as if it is Good and fast, it cannot be cheap, if its fast and cheap, it is unlikely to be good and if it is cheap and good, then it will take a long time to complete. The Project Manager's job is to ensure that all three happen. The product must be Good, Fast and under budget. One of the common 'compromises' a Project manager is often asked to make is to cut corners on some 'time consuming' processes like reviews, test plan writing, documentation of project plan etc. There is a school of thought that these are not required for software development and the only items which are required are code and user manual which are the ultimate deliverables. These perceived short cuts eventually turn out to be long cuts as 'the project will eventually run into trouble at some point for taking the short cuts. The project manager must have extremely good negotiating skills to convince people of the benefits of being process oriented. However, process orientation should not turn into bureaucracy - If bureaucratic approach is taken, the project will get delayed and delayed and will eventually die a natural death.

9.4. Systems Integration Issues :

We are living in a world where technology is changing very rapidly. One of the items where a lot of time is spent is trying out new tools, finding and evaluating compatibility of tools, negotiating and getting trained on the selected tools etc. Technology has become so fragmented that most software involve some pieces give by different vendors which have to talk to each other. The newer the technology, the more complex are the issues relating to system integration as knowledge level on the new tools is not very high. In a project plan one must provide adequate

cushion for these activities. Otherwise, these activities eat into other planned activities which then have to be rushed through. This increase the probability of errors as the chance of introducing an error becomes very high. Once a planned time has been provided for these activities, time spent on these activities can be monitored and ensured that more than budgeted time is not spent - or atleast we are aware that this is likely to lead to a delay in delivery.

9.5. Idealism v/s Realism :

A Process oriented Project manger is a good project manager but a bureaucratic project manager is a lousy project manager. If the clause numbers and policies are rattled for every event in the project, then people associated with the project so get frustrated. Instead of quoting clause number and saying that we have to follow the book, a good project manager will explain the rationale of the process and the benefits of following the process. If this logical approach is taken, it benefits all concerned as people like to be convinced rather than told that this is the way to do things. A project manager must always be open to new ideas and suggestions. The process manual need not be perfect in all respects. If somebody comes up with a good suggestion which is logical and does not result in any loss of control or productivity, it must be evaluated and implemented quickly. The Software Engineering group must also be immediately notified so that they can evaluate it and see if it can be implemented at the organization level.

9.6. Sub-Contractor Management :

If an organization gives out a part of a project to a subcontractor, then the subcontracted project must be monitored as a separate project using all the techniques described so far. The subcontracted project must be linked to the main project, as it will appear as an activity in the main project. Project Management and Quality of deliverables are still very much the responsibility of the parent organization as they have a commitment of overall delivery to the client.

Questions

Very Short Questions:

1. What do Project Management ?
2. What is COCOMO Model ?
3. What is Project Cost estimation ?
4. Define Risk.
5. What is Resource Management ?

Short Questions :

1. Describe Project sizing and Effort estimation.
2. What are the factors of project cost estimation ?
3. What is risk analysis ? Describe the types of Risk involved in project ?
4. What do you mean by Software Project scheduling ?
5. What is software resource planning ?

Long Question :

1. What do you mean by project management.
2. What is risk analysis? Discuss all type of risks for risk analysis.
3. Short note on -
   (a) Line-of-code (b) COCOMO model
   (c) GANTT Chart (d) PERT/CPM network diagram
4. Discuss various Activity Dependency?
5. Discuss various methodology for project management.

❑❑❑

13

DESIGN CONCEPTS AND MODELS

Objectives

1. Abstraction
2. Refinement
3. Modularity
4. Software Architecture
5. Structural Partitioning
6. Data Structure
7. Information Hiding
8. Design Model
9. Design Documentation

A set of fundamental software design concepts has evolved over the past three decades. Although the degree of interest in each concept has varied over the years, each has stood the test of time. Each provides the software designer with a foundation from which more sophisticated design methods can be applied. Each helps the software engineer to answer the following questions:

- What criteria can be used to partition software into individual components?
- How is function or data structure detail separated from a conceptual representation of the software?
- Are there uniform criteria that define the technical quality of a software design?
- M. A. Jackson once said: "The beginning of wisdom for a software engineer is to recognize the difference between getting a program to work, and getting it right". Fundamental software design concepts provide the necessary framework for "getting it right". They provide the underlying basis for development and evaluation of techniques.

1. Abstraction :

Abstraction consists of focusing on the essential, inherent aspects of an entity and ignoring its accidental properties. In system development, this means focusing on what an object is and does, before deciding how it should be implemented. Use

of abstraction preserves the freedom to make decisions as long as possible by avoiding premature commitments to details. Use of abstraction during analysis means dealing only with application-domain concepts, not making design and implementation decision before the problem is understood. Proper use of abstraction allows the same model to be used for analysis, high-level design, program structure, database structure and documentation.

Each step in the software engineering process is a refinement in the level of abstraction of the software solution. During system engineering, software is allocated as an element of a computer based system. During software requirement analysis, the software solution is stated in terms that are familiar in the problem environment. As we move through the design process, the level of abstraction is reduced. Finally, the lowest level of abstraction is reached when source code is generated.

Three widely used abstraction mechanisms in software design are functional/ procedural abstraction, data abstraction and control abstraction. A functional/ procedural abstraction is a named sequence of instructions that has a specific and limited function. Functional/procedural abstraction can be generalized to collections of subprograms. For example, the word "open" on the door. "Open" implies a long sequence of procedural steps (e.g. walk to the door, reach out and grasp knob, turn knob and pull door, step away from moving door, etc.)

A data abstraction is a named collection of data that describes a data object. In the context of the procedural abstraction open noted above, we can define a data abstraction called door. Like any data object, the data abstraction for door would encompass a set of attributes that describe the door (e.g. door type, swing direction, opening mechanism, weight, dimensions). It follows that the procedural abstraction "open" would make use of information contained in the attributes of the data abstraction "door".

Control abstraction is the third form of abstraction used in software design. Like procedural and data abstraction, control abstraction implies a program control mechanism without specifying internal details. Control abstraction is used to state a desired effect without stating the exact mechanism of control.

2. Refinement :

Step-wise refinement is a top-down design strategy originally proposed by Niklaus Wirth. The architecture of a program is developed by successively refining levels of procedural details.

Refinement is actually a process of elaboration. We begin with a statement of function (or description of information) that is defined at a high level of abstraction. That is, the statement describes function or information conceptually, but provides no information about the internal workings of the function or the internal structure of the information. Refinement causes the designer to elaborate on the original statement, providing more and more detail as each successive refinement (elaboration) occurs.

Abstraction and refinement are complementary concepts. Abstraction enables designer to specify procedure and data and yet suppress low-level details. Refinement helps the designer to reveal low level details as design progresses. Both concepts aid the designer in creating a complete design model as the design evolves.

3. Modularity :

The concept of modularity in computer software has been espoused for almost four decades. Software architecture embodies modularity; that is, software is divided into separately named and addressable components, called modules, that are integrated to satisfy problem requirements. Modularity is the single attribute of software that allows a program to be intellectually manageable. Modular system consist of well-defined, manageable units with well-defined interfaces among the units.

Under modularity or over modularity should be avoided. An important question arises when modularity is considered? How do we define an appropriate module of given size? The answer lies in the method(s) used to define module' within a system. Meyer defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system:

- **Modular decomposability** - If a design method provides a systematic mechanism for decomposing the problem into subproblems, it will reduce the complexity of the overall problem, thereby achieving an effective modular solution.
- **Modular composability** - If a design method enables existing (reusable) design components to be assembled into a new system, it will yield a modular solution that does not reinvent the wheel.
- **Modular understandability** - If a module can be understood as a stand-alone unit (without reference to other modules) it will be easier to build and easier to change.
- **Modular continuity** - If small changes to the system requirements result in changes to individual modules, rather than system-wide changes, the impact of change-induced side effects will be minimized.
- **Modular protection** - If an aberrant condition occurs within a module and its effects are constrained within that module, the impact of error-induced side effects will be minimized.

Modularity enhances design clarity, which in turn eases implementation, debugging, testing, documenting and maintenance of the software product.

4. Software Architecture :

Software architecture refers to the overall structure of the software and the ways in which that structure provides conceptual integrity for a system. In its simplest form, architecture is the hierarchical structure of program components (modules), the manner in which these component interact, and the structure of the data that are used by the components.

One goal of software design is to derive an architectural rendering of a system. This rendering serves as a framework from which more detailed design activities are conducted. A set of architectural patterns enables a software engineer to reuse design-level concepts.

Shaw and Garlan describes a set of properties that should be specified as part of an architectural design:

- **Structural properties** - This aspect of the architectural design representation defines the components of system (e.g. modules, objects, filters) and the manner in which those components are packaged and interact with one another.
- **Extra-functional properties** - The architectural design description should address how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability and other system characteristics.
- **Families of related systems** - The architectural design should draw upon repeatable patterns that are commonly encountered in the design of families of similar systems. In essence, the design should have the ability to reuse architectural building blocks.

Given the specification of these properties, the architectural design can be represented using one or more of a number of different models. Structural models represent architecture as an organized collection of program components. Framework models increase the level of design abstraction by attempting to identify repeatable architectural design frameworks (patterns) that are encountered in similar types of applications. Dynamic models address the behavioral aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events. Process models focus on the design of the business or technical process that the system must accommodate. Finally, functional models can be used to represent the functional hierarchy of a system.

Control Hierarchy : Control hierarchy, also called program structure, represents the organization of program components (modules) and implies a hierarchy of control. It does not represent procedural aspects of software such as sequence of processes, occurrence/order of decisions or repetition of operations.

Many different notations are used to represent control hierarchy. The most likely is the tree-like diagram shown below. However, other notations, such as Warnier-Orr and Jackson diagram may also be used with equal effectiveness.

In the figure, dept and width provide an indication of the number of levels of control and overall span of control, respectively. Fan-out is a measure of the number of modules that are directly controlled by another module. Fan-in indicates how many modules directly control a given module.

The control relationship among modules is expressed in the following ways:

A module that controls another module is said to be super ordinate to it; conversely, a module controlled by another is said to be subordinate to the controller.

For example, as shown in figure, module M is super ordinate to modules a, band c. Module h is subordinate to module e and is ultimately subordinate to module M. Width-oriented relationships (e.g. between modules d and e), although possible to express in practice, need not be defined with explicit terminology.

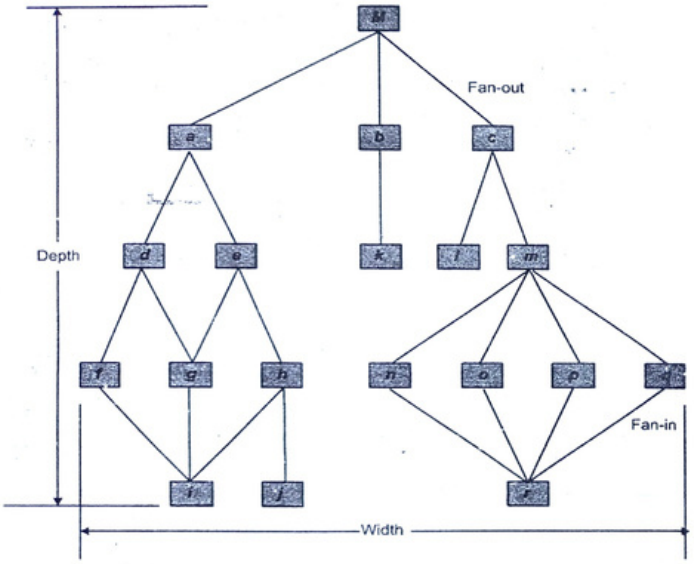


Fig. 1 : Structure terminology

The control hierarchy also represents two subtly different characteristics of the software architecture: visibility and connectivity. Visibility indicates the set of program components that may be invoked or used as data by a given component, even when this is accomplished indirectly. For example, a module in an object-oriented system may have access to a wide array of data attributes that it has inherited, but only make use of a small number of these data attributes. Connectivity indicates the set of components that are directly invoked or used as data by a given component. For example, a module that directly causes another module to begin execution is connected to it.

5. Structural Partitioning :

The program structure should be partitioned both horizontally and vertically.

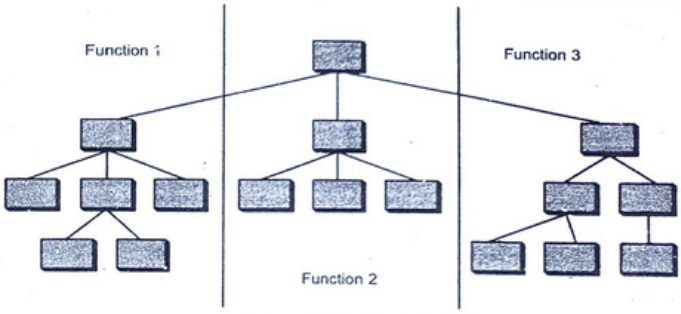


Fig. 2 - Horizontal partitioning

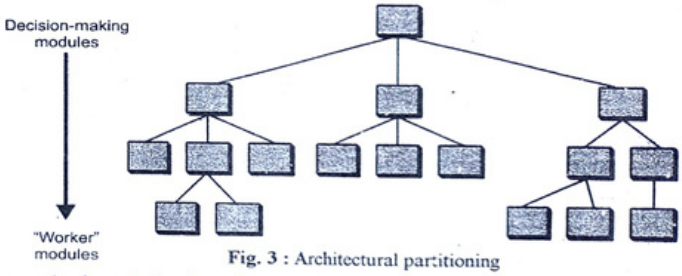


Fig. 3 : Architectural partitioning

As shown in the above figure, horizontal partitioning defines separate branches of the modular hierarchy for each major program function. Control modules, represented by a darker shade, are used to coordinate communication between and execution of program functions. The simplest approach to horizontal partitioning defines three partitions - input, data transformation (often called processing) and output. Partitioning the architecture horizontally provides a number of distinct benefits:

- Results in software that is easier to test
- Leads to software that is easier to maintain
- Results in propagation of fewer side effects
- Results in software that is easier to extend

Because major functions are decoupled from one another, change tends to be less complex and extensions to the system (a common occurrence) tend to be easier

mplish without side effects. On the negative side, horizontal partitioning often es more data to be passed across module interfaces and can complicate the control of program flow (if processing requires rapid movement from one tion to another).

rtical partitioning, often called factoring, suggests that control (decision making) work should be distributed top-down in the program architecture. Top-level ules should perform control functions and do little actual processing work. ules that reside low in the architecture should be the workers, performing all t, computational and output tasks.

The nature of change in program architectures justifies the need for vertical itioning. A change in a control module (high in architecture) will have a higher bility of propagating side effects to \ modules that are subordinate to it. A ge to a worker module, given its low level in the structure, is less likely to the propagation of side effects. In general, changes to computer programs lve around changes to input, computation or transformation and output. The ntrol structure of the program (i.e., its basic behavior) is far less likely to ge. For this reason vertically partitioned architectures are less likely to le to side effects when changes are made and will therefore be more ntainable - a key quality factor.

Structure :

Data structure is a representation of the logical relationship among individual ems of data. Because the structure of information will invariably affect the procedural design, data structure is an important as program structure to the esentation of software architecture.

structure dictates the organization, methods of access, degree of ciativity and processing alternatives for information. It is important to understand c methods available for organizing information and the concepts that rlie information hierarchies.

organization and complexity of a data structure are limited only by the nuity of the designer. There are, however, a limited number of classic data tures that form the building blocks for more sophisticated structures.

A alar item is the simplest of all data structures. As its name implies, a scalar represents a single element of information that may be addressed by an ifier that is, access may be achieved by specifying a single address in storage.

When scalar items are organized as a list or contiguous group, a sequential formed. Vectors are the most common of all data structures and open the to variable indexing of information.

When the sequential vector is extended to two, three and ultimately, an arbitrary ber of dimensions, an n-dimensional space is created. The most common n- sional space is the two-dimensional matrix. In most programming languages, dimensional space is called an array.

Items, vectors and spaces may be organized in a variety of formats. A linked list is a data structure that organizes noncontiguous scalar items, vectors or spaces in a manner (called nodes) that enables them to be processed as a list. Each node contains the appropriate data organization (e.g., a vector) and one or more pointers that indicate the address in storage of the next node in the list. Nodes may be added at any point in the list by redefining pointers to accommodate the new list entry.

Other data structures incorporate or are constructed using the fundamental data structures described above. For example, a hierarchical data structure is implemented using multi linked lists that contain scalar items, vectors and possibly n-dimensional spaces. A hierarchical structure is commonly encountered in applications that require information categorization and associativity. Categorization implies a grouping of information by some generic category. Associativity implies the ability to associate information from different categories.

It is important to note that data structures, like program structures, can be represented at different levels of abstraction.

Software Procedure : Program structure defines control hierarchy without regard to the sequence of processing and decisions. Software procedure focuses on the processing details of each module individually. Procedure must provide a precise specification of processing, including sequence of events, exact decision points, repetitive operations and even data organization/structure.

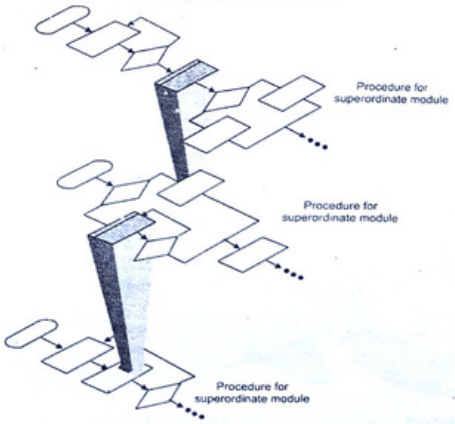


Fig. 4 : Procedure is layered

There is a relationship between structure and procedure. Processing indicated each module must include a reference to all modules subordinate to the module ng described. A procedural representation of software is layered as illustrated in below mentioned figure.

Information Hiding :

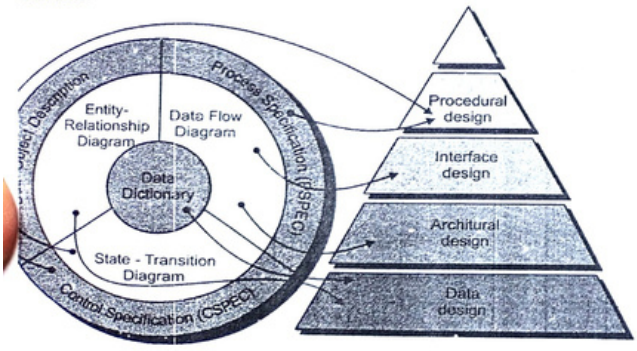
Information hiding is a fundamental design concept for software. The principle nformation hiding suggests that modules should be specified and designed so information (procedure and data) contained within a module is inaccessible to r modules that have no need for such information.

When a software system is designed using the information hiding approach, modules communicate only through well-defined interfaces.

The use of information hiding as a design criterion for modular systems provides reatest benefits when modifications are required during testing and later, during ware maintenance. Because most data and procedure are hidden from other s of the software, inadvertent errors introduced during modification are less y to propagate to other locations within the software.

Design Model :

The design principles and concepts establish a foundation for the creation of lesign model that encompasses representations of data, architecture, interfaces procedures. Like the analysis model, in the design model each of these design esentation is tied to the others, and all can be traced back to software irements.



The analysis model — The design model

Fig. 5 : Translating the analysis model into a software design

The design model is represented as a pyramid. The symbolism of this shape is important. A pyramid is an extremely stable object with a wide base and a low center of gravity. Like the pyramid, we have to create a software design that is stable. By establishing a broad foundation using data design, a stable mid-region with architectural and interface design and a sharp point by applying procedural design, we create a design model that is not easily tipped over by winds of change. A smallest change may cause the pyramid (and the program) to topple.

The methods that lead to the creation of the design model are discussed further. Each method enables the designer to create a stable design that conforms to the fundamental concepts that lead to high-quality software.

9. Design Documentation :

Design specification Outline:

I. Scope
   - A. System objectives
   - B. Major software requirements C. Design constraints, limitations

II. Data Design
   - A. Data objects and resultant data structures
   - B. File and database structures
     1. external file structure
        - a. logical structure
        - b. logical record description
        - c. access method
     2. global data
     3. file and data cross reference

III. Architectural Design
   - A. Review of data and control flow
   - B. Derived program structure

IV. Interface Design
   - A. Human-machine interface design specification
   - B. Human-machine interface design rules
   - C. External interface design
     1. Interfaces to external data
     2. Interfaces to external systems or devices
   - D. Internal interface design rules

V. Procedural Design

For each module:

   - A. Processing narrative
   - B. Interface description
   - C. Design language (or other) description
   - D. Modules used
   - E. Internal data structures
   - F. Comments/restrictions/limitations

VI. Requirements Cross-Reference
VII. Test Provisions
1. Test guidelines
2. Integration strategy
3. Special considerations
VIII. Special Notes
IX. Appendices

The document outlined above can be used as a template for a design specification. Each numbered paragraph addresses different aspects of the design model. The numbered sections of the design specification are completed as the designer refines his or her representation of the software.

The overall scope of the design effort is described above. Much of the information contained in this section is derived from the system specification and the analysis model (software requirements specification).

The data design, describing external file structures, internal data structures and a cross reference that connects data objects to specific files. The architectural design, indicates how the program architecture has been derived from the analysis model. Structure charts are sued to represent the module hierarchy. External and internal program interfaces are represented and a detailed design of the human-machine interface is described. Modules - separately addressable elements of software such as subroutines, functions or procedures - are initially described with an English-language processing narrative explains the procedural function of a module. Later, a procedural design tool is used to translate the narrative into a structured description.

The design specification contains a requirements cross-reference. The purpose of this cross-reference matrix is - to establish that all requirements are satisfied by the software design and to indicate which modules are critical to the implementation of specific requirements.

The first stage in the development of test documentation is contained in the design document. Once software structure and interfaces have been established, we can develop guidelines for testing of individual modules and integration of the entire package. In some cases, a detailed specification of test procedure occurs in parallel with design. In such cases, this section may be deleted from the design specification.

Design constraints, such as physical memory limitations or the necessity for a specialized external interface, may dictate special requirements for assembling or packaging of software. Special considerations caused by the necessity for program overlay, virtual memory management, high-speed processing, or other factors may cause modification in design derived from information flow or structure.

Algorithms descriptions, alternative procedures, tabular data, excerpts from other documents and other relevant information are presented as a special note or as a separate appendix.

Questions

Very Short Questions:

1. What is Abstraction ?
2. Define Modularity ?
3. What is Information hiding ?
4. What do you mean by Software Architecture?
5. What is design model ?

Short Questions:

1. What do you mean by refinment ?
2. What is modulaerity ? Describe Various types of modularity ?
3. What do you mean by Program Structure ?
4. Describe structural Partitioning.
5. Describe Data structure.

Long Question :

1. Short note on -
   (a) Abstraction (b) Refinement (c) Modularity
2. What do you mean by software Architecture?
3. What is structured partitioning? Discuss in detail.
4. What do you mean by design model & design documentation?

❑❑❑