2.0 UNDERSTAND THE SOFTWARE PROJECT MANAGEMENT

2.1 KNOW THE RESPONSIBILITIES OF A SOFTWARE PROJECT MANAGER:

The main goal of software project management is to enable a group of software developers to work efficiently towards successful completion of the project.

2.1.1 JOB RESPONSIBILITIES OF A SOFTWARE PROJECT MANAGER

Software project managers take the overall responsibility of routing a project to success. Most managers take responsibility for project proposal writing, project cost estimation, scheduling, project staffing, project monitoring and control, software configuration management, risk management, interfacing with clients, managerial report writing and presentations.

We can broadly classify the activities of a project manager into project planning and project monitoring and control activities.

1. **Project Planning:** Project planning involves estimating several characteristics of the project and then planning the project activities based on the estimates made. Project planning is undertaken immediately after the feasibility study and before requirements analysis and specification phase.
2. **Project monitoring and control activities:** The project monitoring and control activities starts once the development activities start. The aim of the project monitoring and control activities is to ensure that the development proceeds as per plan.

2.1.2 SKILLS NECESSARY FOR SOFTWARE PROJECT MANAGEMENT

A theoretical knowledge of different project management techniques is necessary to become a successful project manager. In addition to have a good grip of the latest software project management techniques such as cost estimation, risk management, and configuration management etc., project managers need good communication skills and the ability get work done. Some skills such as tracking and controlling progress of project, customer interaction, managerial presentation and team building are largely acquired through experience.

2.2 KNOW ABOUT SOFTWARE PROJECT PLANNING

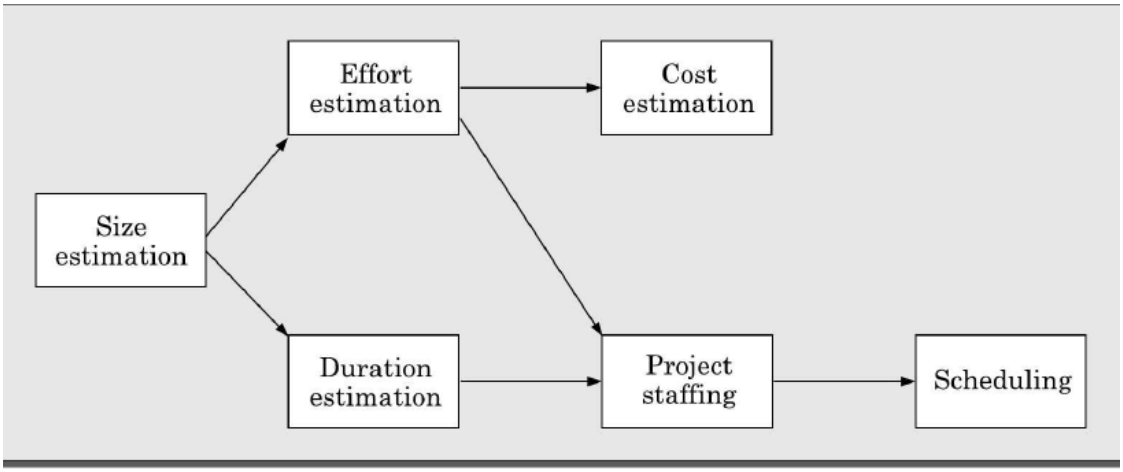
Once a project is found to be feasible, software project managers undertake project planning. Project planning consists of the following essential activities.

1. **Estimation:** The following project attributes have to be estimated.
   a. **Cost:** How much is it going to cost to develop the software?
   b. **Duration:** How long is it going to take to develop the product
   c. **Effort:** How much effort would be required to develop the product.
2. **Scheduling:** After the estimations are made, the schedules for manpower and other resources have to be developed.
3. **Staffing:** Staff organization and staffing plans have to be made.

4. **Risk management:** Risk identification, analysis, and abatement planning have to be made.
5. **Miscellaneous plans:** Several other plans such as quality assurance plan, configuration management plan etc has to be done.

Size is the most fundamental parameter based on which all other estimates are made.

For large project sometimes project managers undertake project planning in stages. This technique of staggered planning is known as sliding window planning. In sliding window technique, starting with an initial plan, the project is planned more accurately in successive development stages.



Precedence ordering among planning activities.

2.2.1 THE SPMP DOCUMENT: Once the project planning is complete, project managers document their plans is a **Software Project Management Plan (SPMP) document**.

Organization of the Software Project Management Plan (SPMP) Document:

1. **Introduction**
    a. Objectives
    b. Major Functions
    c. Performance Issues
    d. Management and Technical constraints
2. **Project Estimates**
    a. Historical Data used
    b. Estimation Techniques used
    c. Effort, Resources, Cost and Project Duration Estimates
3. **Schedule**
    a. Work Breakdown structure
    b. Task Network Representation
    c. Gantt chart representation
    d. PERT Chart representation

4. **Project Resources**
    a. People
    b. Hardware and Software
    c. Special Resources
5. **Staff Organization**
    a. Team Structure
    b. Management Reporting
6. **Risk Management Plan**
    a. Risk Analysis
    b. Risk identification
    c. Risk Estimation
    d. Risk abatement procedure
7. **Project tracking and control plan**
8. **Miscellaneous Plans**
    a. Process Tailoring
    b. Quality Assurance Plan
    c. Configuration Management plan
    d. Validation and Verification
    e. System Testing Plan
    f. Delivery, Installation and Maintenance Plan.

2.3 STATE THE METRICS FOR PROJECT SIZE ESTIMATION

Accurate estimation of the problem size is fundamental to satisfactory estimation of other project parameters such as effort, time and Cost of Project. The project size is the measure of the problem complexity in terms of the effort and time required to develop the product.

Currently, two metrics are popularly used to estimate size: Line of Code (LOC) and Function Point(FP).

2.3.1 Lines of Code:

LOC is the simplest among all metrics available this metric is extremely popular. LOC measures the size of a project by counting the number of source instructions in the developed program. However lines used for commenting the code and header lines are ignored.

Determining the LOC count at the end of the project is very simple. However, accurate estimation of the LOC count at the beginning of a project is very difficult.

LOC given problem size vary widely with individual coding style.

LOC is a measure of the coding activity alone. A good problem size measure should consider the effort needed to design, code, test etc.,

LOC measure correlate poorly with the quality and efficiency of the code.

LOC metric penalizes use of higher-level programming languages, code reuse.

It is very difficult to accurately estimate LOC in the final product from the problem specification.

2.3.2 Function Point Metric

One of the advantages of function point metric is that it can be used to easily estimate the size of the software directly from the problem specification.

The idea behind Function point metric is that the size of a software product depends on the number of different functions or features it supports. Software supporting many features would be large size than a product with less features.

A computation of number of input and output data values to a system gives some indication of the number of functions supported by the system. In addition to the number of basic functions that software performs, the size is also dependent on the number of files and the number of interfaces.

Function point metric computes the size of software in units of function points or FPs. Function point is computed in three steps. The first step is to compute the unadjusted function point(UFP). In the next step, the UFP is refined to reflect the differences in the complexities. In the final step, FP is computed by further refining UFP.

UFP=(Number of inputs)*4 + (Number of Outputs)*5 + (number of inquiries)*4 + (number of files)*10 + (Number of Interfaces)*10.

The computed UFP is refined in the next step. The complexity level of each of the parameters is graded as simple, average, or complex.

Type	Simple	Average	Complex
Input(I)	3	4	6
Output(O)	4	5	7
Inquiry(E)	3	4	6
Number of Files(F)	7	10	15
Number of Interfaces	5	7	10

Rather than Input being computed as four function points, very simple input can be computed as three function points and very complex input as six function points.

A Technical Complexity Factor(TCF) for the project is computed as the TCF is multiplied with UTP to yield FP. The TCF expresses the overall impact of various project parameters that can influence the development effort such as high transaction rate, response time requirements, scope or reuse etc. Albrecht identified 14 parameters than can influence the development effort. Each of these 14 factors is assigned a value from 0 to 6. The resulting number are summed,

yielding the total degree of influence (DI). TCF is computed as (0.65+0.01*DI). As DI can vary from 0 to 84, TCF can vary from 0.65 to 1.35. Finally, FP is given as FP=UTP * TCF.

2.4 EXPLAIN THE THREE PROJECT ESTIMATION TECHNIQUES

Estimation of various project parameters is a basic project planning activity. The important project parameters that are estimated include: Project size, effort, Project duration and Cost.

There are three broad categories of Estimation Techniques

- Empirical Estimation Technique
- Heuristic Technique
- Analytical Estimation Technique

2.4.1 Empirical Estimation Technique

Empirical Estimation technique is based on making an educated guess of the project parameters. While using this Technique, prior experience with development of similar product is helpful.

Expert Judgment Technique: In this Technique, an expert makes an educated guess of the problem size after analyzing the problem thoroughly. This technique is subject to human error and individual bias, the expert may overlook some parameters, he may not have experience and knowledge on all aspects of project.
A more refined form of expert judgment is the estimation made by a group of experts.

Delphi Cost Estimation: Estimation is carried out by a team comprising of a group of experts and a coordinator. The coordinator provides each estimator with a copy of SRS document and a form to record his estimate. The coordinator prepares the summary of the responses of all estimators the prepared summary is distributed to the estimators. Based on the summary the estimator re-estimates.

2.4.2 Heuristic Technique: Heuristic technique assumes that the relationship among different project parameters can be modeled using suitable mathematical expressions. Different Heuristic estimation models can be divided into two classes: single variable model and Multivariable model.

A single variable model takes the following form:

$$\text{Estimated parameter} = c1 * e^{d1}$$

'e' is a characteristic of the software which has already being estimated. Estimated parameter could be effort, duration, staff size etc. c1 and d1 are constants are usually determined using data collected from past projects. The basic COCOMO model is an example of single variable cost estimation Model.

A Multivariable cost estimation model takes the following form:

$$\text{Estimated resource} = c1 * ep1^{d1} + c2 * ep2^{d2} + \ldots$$

Where ep1, ep2,.. are basic characteristics of software already estimated and c1,c2,d1,d2.. are constants determined from historical data. The intermediate COCOMO Model is an example of Multivariable Model.

COCOMO – A Heuristic Estimation Technique:

COCOMO (COnstructive COst estimation MOdel) was proposed by Boehm, According to him any software project can be classified into 3 categories based on complexity: organic, semidetached, and embedded.

Organic: Project is organic type, if the project deal with developing a well –understood application program.

Semidetached: If the team consists of a mixture of experienced and inexperienced staff.

Embedded: If the software being developed is strongly coupled to complex hardware.

Basic COCOMO Model: It gives an approximate estimates of the project parameters.

$$\text{Effort} = a1 \times (\text{KLOC})^{a2} \text{ PM}$$

$$\text{Tdev} = b1 \times (\text{Effort})^{b2} \text{ Months}$$

KLOC is the estimated size of the product expressed in Kilo Lines of Code.

a1,a2,b1,b2 are constants for each category of product

Tdev is the estimated time to develop the software expressed in Months.

Effort is the total effort required to develop the software product, expressed in Person Months (PMs).

Estimation of development Effort

Organic : Effort = $2.4(\text{KLOC})^{1.05}$ PM

Semidetached : Effort = $3.0(\text{KLOC})^{1.12}$ PM

Embedded : Effort = $3.6(\text{KLOC})^{1.20}$ PM

Estimation of development Effort

Organic : Tdev = $2.5(\text{Effort})^{0.38}$ Months

Semidetached : Tdev = $2.5(\text{Effort})^{0.35}$ Months

Embedded : Tdev = $2.5(\text{Effort})^{0.32}$ Months

Intermediate COCOMO: The Basic COCOMO assumes that effort and Time are functions of product size alone. To obtain accurate estimation of effort and duration, the effect of all relevant parameters must be taken into account. The intermediate COCOMO model recognize this fact and refines the initial estimates obtained using Basic COCOMO by using a set of 15 cost drivers.

Boehm requires the project manager to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these rating, he suggested appropriate cost driver value which should be multiplied with the initial estimate obtained using the Basic COCOMO.

2.4.3 Analytical Estimation Technique:

Analytical estimation Technique derive the required results starting with certain basic assumptions regarding the project. Halstead's Software science is an example of an analytical technique. Halstead's software science is especially useful for estimating software maintenance efforts.

HALSTEAD's Software Science – An Analytical Technique:

Halstead used a few primitive program parameters to develop the expressions for overall program length, potential minimum volume, actual volume, language level, effort and development time.

For a given program, let η1 be the number of unique operators η2 be the number of unique operands, N1 be the total number of operators used in program and N2 be the total number of operands used in the program.

Length and Vocabulary

$$\textbf{Length } N = N1 + N2$$

$$\textbf{Vocabulary } \eta = \eta 1 + \eta 2$$

Program Volume: The length of program depends on the choice of the operators and operands used

$$V = N \log_2 \eta$$

Potential Minimum Volume: The potential minimum volume V* is defined as the volume of the most brief program in which a problem can be coded.

$$V^* = (2 + \eta 2)\log_2(2 + \eta 2)$$

The program level L is given by $L = V^* / V$

Effort and Time:

$$\textbf{Effort } E = V/L \text{ or } E = V^2/V^* \text{ (Since } L = V^*/V)$$

$$\textbf{Time } T = E/S$$

Where S is the speed of Mental Discrimination its recommended value for programming applications is 18.

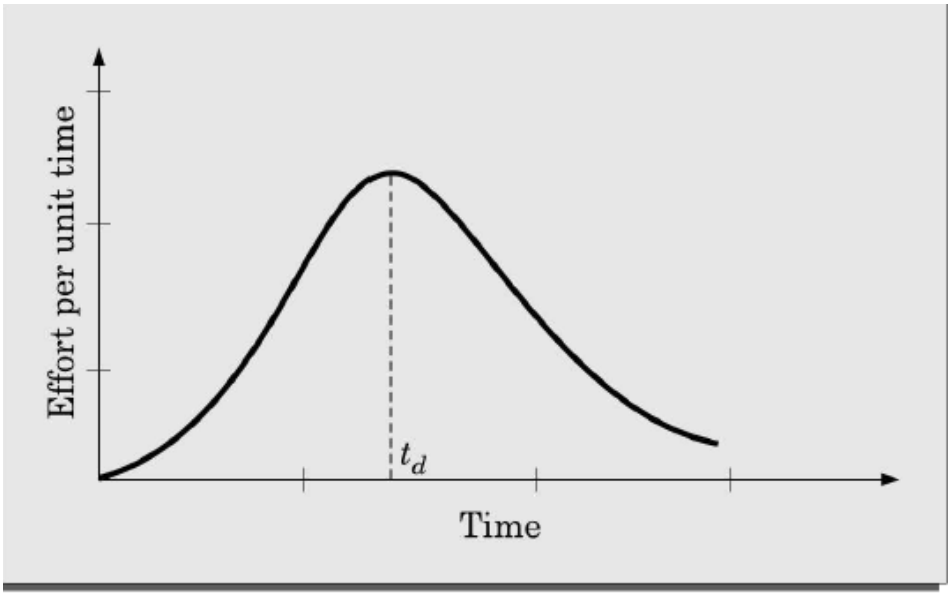
2.5 EXPLAIN THE TWO DIFFERENT WORKS OF STAFFING LEVEL ESTIMATIONS:

Once the effort required to develop software has been determined, it is necessary to determine the staffing requirement for the project.

Putnam was first to study the problem of staffing pattern for software project. He extended the work of Norden who earlier investigated the staffing pattern for Research and Development (R&D) type of projects.

2.5.1 Nordens work: Norden studied the staffing pattern of several R&D projects. He found that the staffing pattern can be approximated by the Rayleigh Distribution Curve. Norden represented the Rayleigh curve by the following Equation.

$$E = \frac{K}{{t_d}^2} * t * e^{\frac{-t^2}{2{t_d}^2}}$$



Rayleigh curve.

Where **E** is an indication of the number of developers at any particular time during the duration of the project. **K** is the area under the curve, and **td** is the time at which the curve attains maximum value. Norden work are applicable to general R&D projects and not meant to model the staffing pattern for software projects.

2.5.2 Putnam's Work:

By analyzing a large number of army projects, Putnam derived the following expression:

$$L = C_k K^{1/3} t_d^{4/3}$$

K is the total effort expanded (in PM) in the product development.

L is the product size in KOLC.

td corresponds to time of system and integration and testing. td is the time required to develop the software.

C_k is the state of technology constant and reflects constraints that impede the progress of the programmer. Typical value of C_k=2 for poor development environment. C_k=8 for good software development environment and C_k = 11 for excellent environment.

Putnum suggested that optimal staff built-up on a project should follow the Rayleigh Curve. Only a small number of developers are needed at the beginning of a project to carry out planning and specification tasks. As the project progresses, the number of developers increases and reaches a peak during product testing. After implementation and unit testing, the number of project staff falls.

Staff built up should not be carried out in large installments. The team size should either be increased or decreased slowly whenever required to match the Rayleigh Norden curve.

It should be clear that a constant Level of manpower throughout the project duration would lead to wastage of effort and increase the time and effort required to develop the product.

If we examine the Rayleigh curve we can see approximately 40% of the area under the curve is to the left of td and 60% area is to the right. The effort required to develop the product to its maintenance effort is approximately in 40:60 ratio.

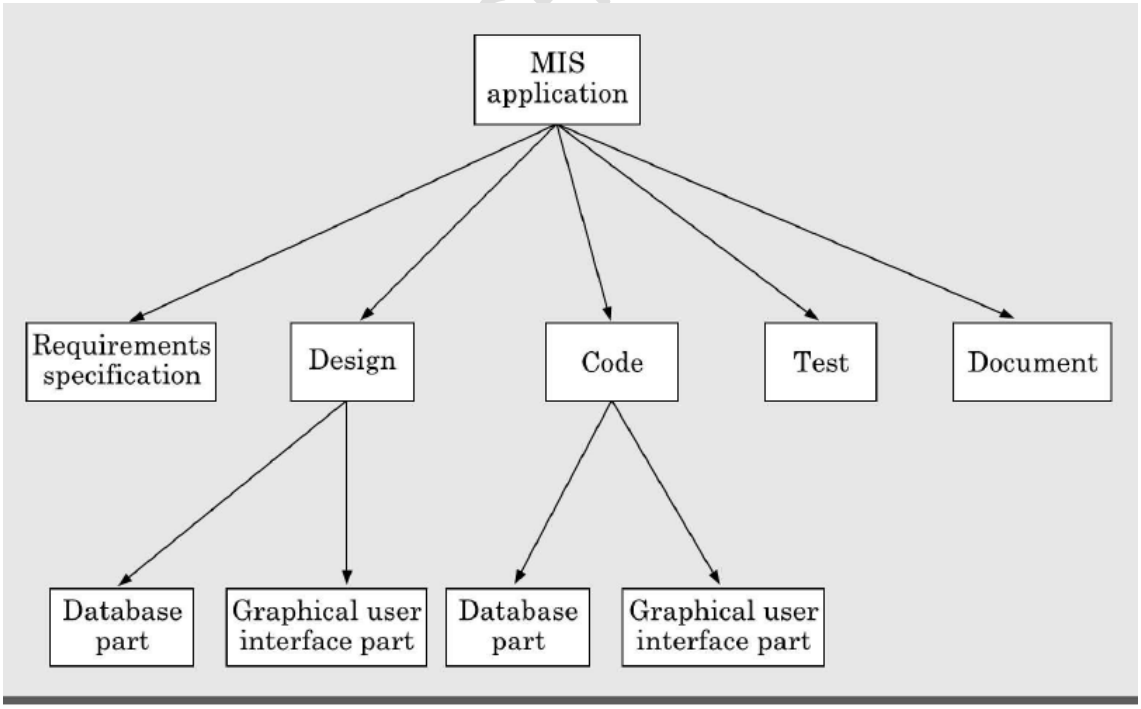
2.6 UNDERSTAND THE FOUR WAYS OF SCHEDULING

Scheduling is an important project planning activity. It involves deciding which tasks would be taken up when. In order to schedule a software project manager needs to do the following.

- Identify all the tasks needed to complete the project.
- Break down large tasks into small activities.
- Determine the dependency among different activities.
- Establish the most likely estimates for the time durations necessary to complete the activities.
- Allocate resources to activities.
- Plan the starting and ending dates for various activities.
- Determine the Critical Path. A critical path is a chain of activities that determine the duration of the project.

2.6.1 WORK BREAK DOWN STRUCTURE

Work Breakdown structure (WBS) is used to decompose a given task set recursively into small activities. WBS provides a notation for representing the major task needed to be carried out in order to solve a problem. The root of the tree is labeled by the problem name. Each node of the tree is broken down into smaller activities that are made the children of the node. Each activity is recursively decomposed into smaller sub activities until at the leaf level, the activities require approximately two weeks to develop.



Work breakdown structure of an MIS problem.

2.6.2 ACTIVITY NETWORKS AND CRITICAL PATH METHOD

WBS representation of a project is transformed into an activity network by representing activities identified in WBS along with their interdependencies. An activity network shows the different activities making up a project, their estimated durations, and interdependencies.

Managers can estimate the time durations of different tasks in several ways.

1) They can empirically assign duration of different tasks.
2) Many managers believe that an aggressive schedule motivates the developers to do a better and faster job.
3) Let each engineer himself estimate the time for an activity he can be assigned to.

Critical Path Method (CPM).

From the activity network representation, following analysis can be made.

The minimum time (MT) to complete the project is the maximum of all paths from start to finish.

The earliest start (ES) time of a task is the maximum of all paths from the start to this task.

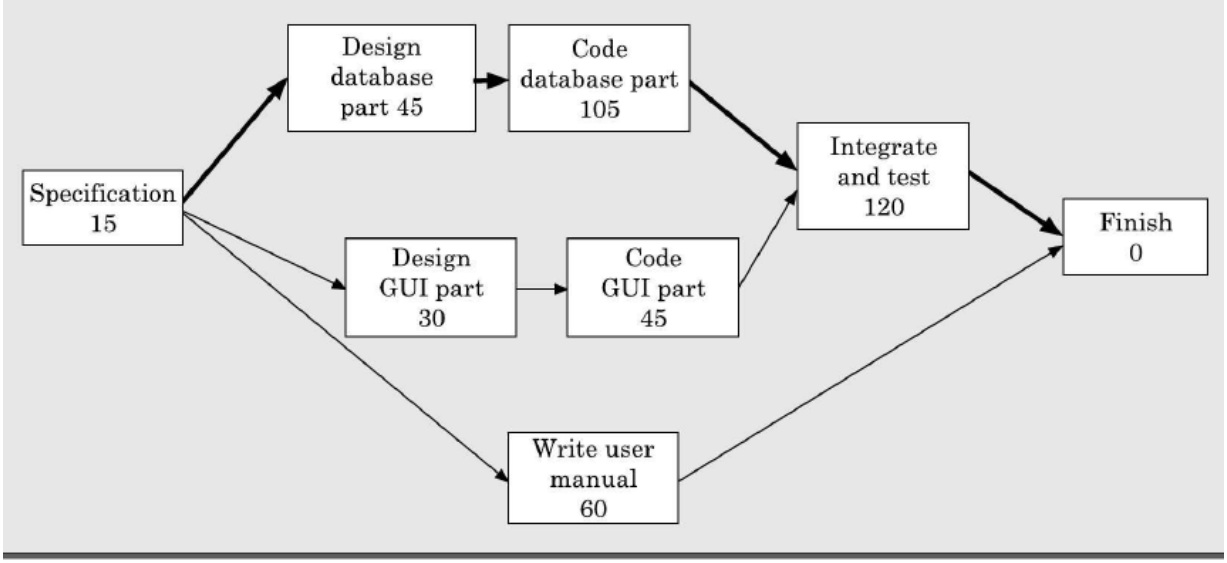
The earliest finish time (EF) of a task is the sum of the earliest start time of the task and the duration of the task.

The Latest start (LS) time is the difference between MT and the maximum of all paths from this task to the finish.

The latest finish (LF) time of a task can be obtained by subtracting maximum of all paths from this task to finish from MT.

The slack time (ST) or float time is LS-EF the total time that a task may be delayed before it will affect the end time of the project.

A critical task is one with a zero slack time. A path from the start node to the finish node containing only critical tasks is called a critical path. Any path whose duration equals MT is a critical path. There can be more than one critical path for a project.

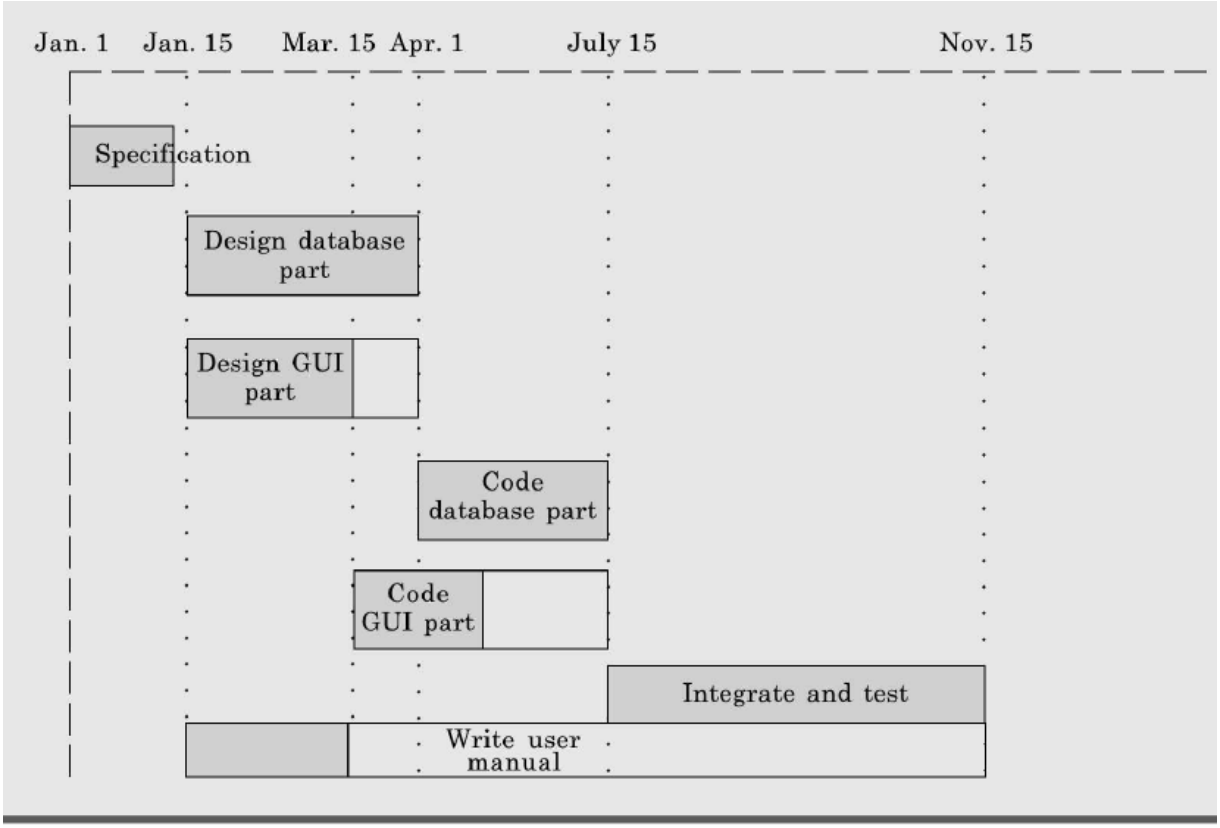


Activity network representation of the MIS problem.

Task	*ES*	*EF*	*LS*	*LF*	*ST*
Specification	0	15	0	15	0
Design Database	15	60	15	60	0
Design GUI part	15	45	90	120	75
Code data base	60	165	60	165	0
Code GUI Part	45	90	120	165	75
Integrate and Test	165	285	165	285	0
Write user Manual	15	75	225	285	210

2.6.3 GANTT CHARTS

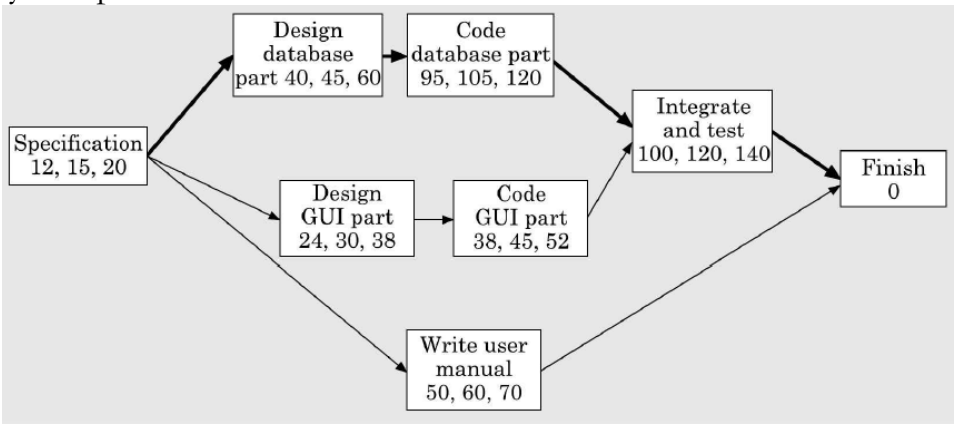
Gantt charts are mainly used to allocate resources to activities. The resources allocated to activities include staff, hardware and software. A gantt chart is a special type of Bar chart where each bar represents an activity. The bars are drawn along a time line. The length of each bar is proportional to the duration of time planned for the corresponding activity.
In the Gantt chart used for software project management, each bar consist of a while part and a shaded part. The shaded part shows the length of time each task is estimated to take. The white part shows the slack time, that is the latest time by which a task must be finished.



Gantt chart representation of the MIS problem.

2.6.4 PERT CHARTS

PERT (Project Evaluation and Review Technique) chart consist of a network of boxes and arrows. In a PERT chart instead of making a simple estimate for each task, pessimistic, likely and optimistic estimates are made. The boxes in PERT charts are usually annotated with the pessimistic, likely and optimistic estimates for a task. This makes the critical path analysis in PERT chart very complex.



PERT charts are more sophisticated form of activity chart. The activity diagram shows only estimated task durations, the actual durations might vary from the estimated durations, the utility of the activity diagrams are limited.
Gantt chart is helpful in planning the utilization of resources. While PERT chart is useful for monitoring the timely progress of activities. Also it is easier to identify parallel activities in a project using a PERT chart.

2.7 LEARN HOW TO DO STAFFING – " WHO IS A GOOD SOFTWARE ENGINEER?"

Software project managers usually take the responsibility of choosing their team. Therefore, they need to identify good software developers for the success of the project. Experiments have revealed that their exist a large variability of productivity between worst and the best software developers in a scale of 1 to 30. Therefore choosing good software developers is crucial to the success of a project.

Who is a Good Software Engineer? Good software developers should posses

1. Exposure to systematic techniques i.e familiarity with software engineering principles.
2. Good Technical knowledge of the project area (domain knowledge)
3. Good Programming abilities.
4. Good Communication skills like oral, written and interpersonal skills.
5. High motivation.
6. Sound knowledge of fundamentals of computer science
7. Intelligence
8. Ability to work in a team.
9. Discipline.

Studies show that these attributes vary as much as 1:30 for poor to bright candidates. An experiment conducted by sackman shows that the ratio of coding hours for the worst to the best programmer is 25:1 and the ration of debugging hours is 28:1.

Technical knowledge in the area of the project is an important factor determining the productivity of an individual and the quality of the product that he develops.

Since software development is a group activity, it is vital for a software developer to possess three main kinds of communication skills: oral, Written and Interpersonal.

Motivation Level of software developer is another crucial factor contributing to his work quality and productivity. Motivation is to a great extent determined by personal, family and social background.

2.9 EXPLAIN RISK MANAGEMENT

A Risk is any expected unfavorable event or circumstance that can occur while a project is underway. If a risk becomes real it can adversely affect the project and slow down the successful and timely completion of the project. Risk management aims at reducing the impact of all kinds

or risks that might affect a project. Risk management consists of three essential activities: risk identification, risk assessment and risk containment.

2.9.1 Risk Identification: The project manager need to predict the risks in the project as early as possible so that the impact of the risk can be minimized by making effective risk management plans.

A project can be affected by a large variety of risks. In order to be able to systematically identify the important risks that might affect the project, it is necessary to categorize risks into different classes. There are three main categories of risks which can affect a software project as follows:

Project Risks: Project risks concern various forms of budgetary, schedule, personnel, resource and customer-related problems. An important project risk is schedule slippage.

Technical Risk: Technical risk concern potential design, implementation, interfacing, testing and maintenance problems. Technical risk also includes ambiguous specification, incomplete specification and change specification.

Business Risk: This type of risk includes risks of building an excellent product that no one wants, losing budgetary or personnel commitments.

2.9.2 Risk Assessment: The objective of risk assessment is to rank the risks in terms of their damage causing potential. For risk assessment, first each risk should be rated in two ways.

1. The likelihood of a risk coming true (r).

2. The consequence of the problem associated with that risk (s)

Based on these two factors, the priority of each risk can be computed.

$$P=r * s$$

Where P is the priority with which the risk must be handled. If all identified risks are prioritized, then the most likely and damaging risks can be handled first.

2.9.3 Risk Containment: After all the identified risks of a project are assessed, plans must be made to first control the most damaging and most likely risks. There are three main strategies to plan for risk containment.

Avoid the Risk: Risks can be avoided in several ways, such as discussing with the customer to change the requirements to reduce the scope of the work, giving incentives to developers to avoid risk of manpower turnover.

Transfer the Risk: This strategy involves getting the risk component developed by a third party, buying insurance cover and so on.

Risk Reduction: This involves planning ways to contain the damage due to a risk. For example, if there is risk that some key personnel might leave, new recruitment may be planned. Building a prototype that tries out pieces of the technology that you are trying to use can address technical risk.

To choose between the different strategies of handling a risk, the project manager must consider the cost of handling the risk and the corresponding reduction of risk. For this we may computer the risk leverage of different risks.

$$Risk\ leverage = \frac{\text{Risk exposure before reduction} - \text{Risk exposure after reduction}}{\text{Cost of reduction}}$$

Important Questions from Previous Questions Paper : Short Answers

1. List any three features of high-level languages. (Mar 2015)
2. List at least three project estimation techniques. (Mar 2015)
3. What is UFP in function Point metric? (Mar 2015)
4. What are the shortcomings of lines of code metric? (Mar 2015)
5. Distinguish between program and software product. (DIT Mar 2015)
6. List the responsibilities of a software project manager. (DIT Mar 2015)
7. Write about Delphi cast estimation. (DIT Mar 2015)
8. What is the disadvantage of programs with respect to software product? (Oct 2015)
9. What is risk exposure? (Oct 2015)
10. List out the different empirical estimation techniques. (Oct 2015)
11. What is the basic difference between control-flow-oriented and dataflow-oriented design techniques?
12. What are the skills necessary for software project management? (Oct 2014)
13. What are the three main strategies used for risk containment? (Oct 2014)
14. What is the solution to the software crisis? (DIT Mar 2014)
15. List the responsibilities of a software project manager. (DIT Oct 2014)
16. What is an SPMP document? (DIT Oct 2014)
17. Who can be considered as good software engineer? (DIT Oct 2014)
18. Write about early computer programming (Mar 2014)
19. Explain the skills necessary for software project management (Mar 2014)
20. Explain how to assess a risk. (Mar 2014)
21. Explain Delphi cost estimation technique. (Mar 2014)
22. What is software crisis? (DIT Mar 2014)
23. What are the skills necessary for software project manager? (DIT Mar 2014)
24. List at least three different methods of scheduling in software development (DIT Mar 2014)

Essay Questions

1. Explain classical waterfall model (Mar 2015)
2. Explain briefly about critical path method (Mar 2015)
3. Explain classical waterfall model with a neat diagram (DIT Mar 2015)
4. Explain the following : 4+6 *(a)* Line of code *(b)* PERT charts (DIT Mar 2015) \
5. Explain about classical waterfall model with neat diagram (Oct 2015)
6. Explain about PERT chart (Oct 2015)
7. Explain briefly about function point metric (Oct 2015)
8. Explain Spiral model of software development with a neat diagram. (Oct 2014)
9. *(a)* List the project activities chronologically; a software project manager needs to do for scheduling. 5
   (b) Explain the critical path method. (Oct 2014)
10. *(a)* What are the attributes that a good software engineer should possess? 5
    (b) Explain the empirical estimation techniques. 5 (Oct 2014)
11. Explain about software life cycle models in detail ((DIT Oct 2014)
12. Explain about metrics for project size estimation (DIT Oct 2014)
13. Explain about staffing level estimation in detail. (DIT Oct 2014)
14. *(a)* Compare and contrast between programs and software products.
    (b) Discuss iterative waterfall model with (Mar 2014)
15. *(a)* Explain the function point metric used for project size estimation.
    (b) Explain briefly about heuristic techniques (Mar 2014)
16. Explain PERT chart with a neat sketch (Mar 2014)
17. Explain iterative waterfall model. (DIT Mar 2014)
18. Explain lines of code in project estimation. (DIT Mar 2014)
19. Explain risk management, risk identification and risk assessment (DIT Mar 2014)