APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

STUDY MATERIALS

KTU ASSIST

a complete app for ktu students

Get it on Google Play

www.ktuassist.in

Module 3

Planning and Design phase

SOFTWARE(S/W) PROJECT PLANNING

Ques 1) What do you understand by planning phase of software development?

Or

What is software planning?

Or

What is project plan? What are the different types of project plan?

Ans: Software Project Planning

Software planning starts with the statement of work which is to be accomplished and other limitations and objectives that describe the software project.

The software planning process comprises of certain steps like determining the size of the software product and resources required, fixing a schedule, recognising and assessing risks, and negotiating commitments. These steps are essential for establishing the software development plan.

According to Project Management Body of Knowledge, "A project plan is a formal, approved document used to guide both project execution and project control. The primary uses of the project plan are to document planning assumptions and decisions, facilitate communication among stakeholders, and document approved scope, cost, and schedule baselines. A project plan may be summarised or detailed."

According to PRINCE (PRojects IN Controlled Environments),"A statement of how and when a project's objectives are to be achieved, by showing the major products, milestones, activities and resources required on the project."

Figure 3.1 shows the process, communications and other aspects of the project:

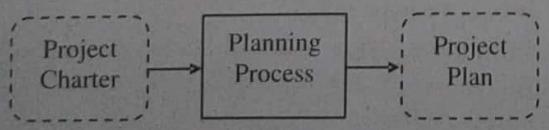


Figure 3.1: Project Planning

The software development plan develops the foundation for performing and managing the activities of the software project. It also fulfils the commitments made to the software project's customer according to the availability of resources, limitations, and abilities.

A project has to be divided into various separate activities which might be pursued in parallel or series, throughout the entire planning stage. The questions which may arise in this situation are:

1) How much effort is needed to accomplish each activity?
2) How much time is required?
3) What will be the cost of the complete project?

The **project plan** describes the strategy which will be used to make the project more effective and executable. This plan describes the work that has to be carried out on the project and who will do it. It comprises the following:

1) A Statement of Work (SOW) which explains all the work that is to be performed along with list of people by whom the work will be completed.
2) A list of resources which comprises of resources required for product and their accessibility.
3) A work breakdown structure with estimations.
4) A plan of the project.
5) A risk plan that recognises risks and also specifies how they are to be handled if faced with them.

A project plan helps in controlling of the project as it puts down the goals, expectations and terminology between the stakeholders and the participants. For a project to be successful there must be coordination among all members.

A starting point is established to measure progress. When the projects starts to diverge from the plan devised at the starting, then efforts can be made to keep the project back on track.

The project team goes through the project plan which helps the team to coordinate amongst each other and rehearse the development of the project. They assess risks, development life cycles, resource availability, budgets and important tasks. They will also face problems and solve them to get a feel of the work before real project development. Having a plan does not mean proper planning. Planning is a continuous process which helps to incorporate alterations in the project in a systematic way.

Types of Project Plan

1) **Software Development Plan:** This is the central plan that decides the way in which system has to be developed.
2) **Quality Assurance Plan:** This identifies the quality procedures and criterions which are to be applied.
3) **Validation Plan:** This delineates how a client should authenticate a system that has been developed.

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

4) **Configuration Management Plan:** This describes the designing and installing of the system.

5) **Maintenance Plan:** This describes the ways of maintaining the system.

6) **Staff Development Plan:** This defines how applicants' skills are to be developed.

Ques 2) What are the characteristics of project planning? What are the objectives of project planning?

Ans: Characteristics of Project Planning

Project planning has five basic characteristics as depicted in **figure 3.2**:

1) **Scope:** Recognises the project goals and things required to attain those goals.

2) **Resource:** These helps in attaining the goals.

3) **Time:** When tasks should be performed to complete the goal.

4) **Quality:** This is the standard that acts as the benchmark of a project.

5) **Risk:** These includes list of possible problems and solutions in case of their occurrence.

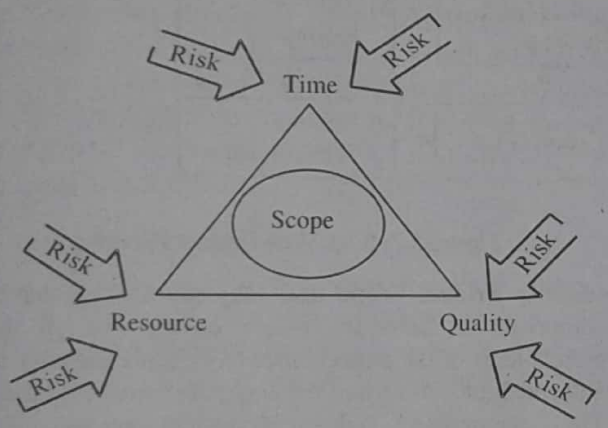


Figure 3.2: Characteristics of Software Project Planning

Project Planning Objectives

The main objectives of project planning are as follows:

1) Understanding problem scope,
2) Making use of historical data,
3) Execute functional disintegration,
4) Estimation of effort and/or function and/or size,
5) Carry out risk analysis,
6) Developing a work flow structure, and
7) Defining a project schedule.

Ques 3) Discuss the different sections of software project management plan.

Ans: Structure of a Software Project Management Plan

The project management plan comprises of four major sections:

1) **Project Summary:** This contains the gist of the project explaining the objectives of the project from the customer's as well as the company's perspective. This provides clarity to all parties involved as to why the project is being undertaken. It has information pertaining to project's beginning and finish dates, the project leader, customer agreements, project goals, main commitments made to the customer on milestones and deliverables. Apart from these, assumptions are also scheduled as they provide an idea about risks. Explanations on billing details are also given.

2) **Project Planning:** The results of implementing different project planning procedures are mentioned in the project planning section. This describes the development process being used, tailoring notes, requirement change management process, requirement traceability plans, effort and plan estimates with their origin. It also mentions the individual's needs by either skill, role, month-wise, or combining all these. Besides, it also defines the development environment required, the tools engaged and any project-specific training plans. This section also mentions the quality and risk management plans.

3) **Project Tracking:** The measurements that are to be taken and the systems to be used for data recording, project tracking activities, occurrence and type of the progress reporting and escalation procedures are described in the project tracking section.

4) **Project Team:** This mentions the project team with its structure and also the roles and responsibilities of the different people involved.

Ques 4) Discuss the major issues in planning a software project.

Ans: Major Issues in Planning a Software Project

There are certain major concerns while planning a software engineering project:

1) Software requirements are often incorrectly and incompletely mentioned.

2) Certain software specifications are unsteady and require frequent and significant changes.

3) Generally, planning is ignored because of the perception that plans are a waste and will need to change midway.

4) Scheduling cost and schedule are based on marketing requirements and are not updated on system needs.

5) A genuine cost and schedule estimate cannot be made as it is not easy to predict the size and complexity of a software project.

6) Cost and schedules are not re-calculated with changes in either needs or development environment.

7) Risk factors are not reviewed or controlled.

8) Project data of past projects are not gathered by majority of software development organisations.

9) Companies do not have a software development policy.

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

Ques 5) Explain the software scope in detail.

Ans: Software Scope

The first activity in software project planning is the determination of software scope. Software scope describes:

1) Functions and features that are to be delivered to end-users
2) Data that are input and output
3) "Content" presented to users when using the software
4) Performance, constraints, interfaces and reliability requirements.

Scope is defined using techniques like:

1) A narrative description of software scope based on communication with all stakeholders
2) A set of scenarios or Use Cases developed by end-users.

For understanding the scope, one must understand the:

1) Customer needs
2) Business context
3) Project boundaries
4) Customer's motivation
5) Likely reasons for change

Once scope has been identified (with the concurrence of the customer), it is reasonable to ask:

1) Can we build software to meet this scope?
2) Is the project feasible?

All too often, software engineers rush past these questions (or are pushed past them by impatient managers or customers), only to become mired in a project that is doomed from the onset.

Putnam and **Myers** correctly suggest that scoping is not enough. Once scope is understood, the software team and others must work to determine if it can be done within the dimensions just noted.

Ques 6) What are the different activities of project planning?

Ans: Project Planning Activities

Figure 3.3 provides an outline of the main planning activities:

The **steps** of planning activities are as below:

1) **Select Project:** The process of project selection is used to evaluate the various project ideas and choose the one that has the utmost importance. Thus only a brief explanation of the project is taken into account to select the project because it is in the form of just suggestions at this stage. The following criteria are considered while selecting the project:
   i) **Benefits:** This is an estimation of the positive results of the project. It defines why the project is being commenced.
   ii) **Feasibility:** It estimates the probability of the success of the project. That is, it is a measure of degree to which the objectives will be attained.

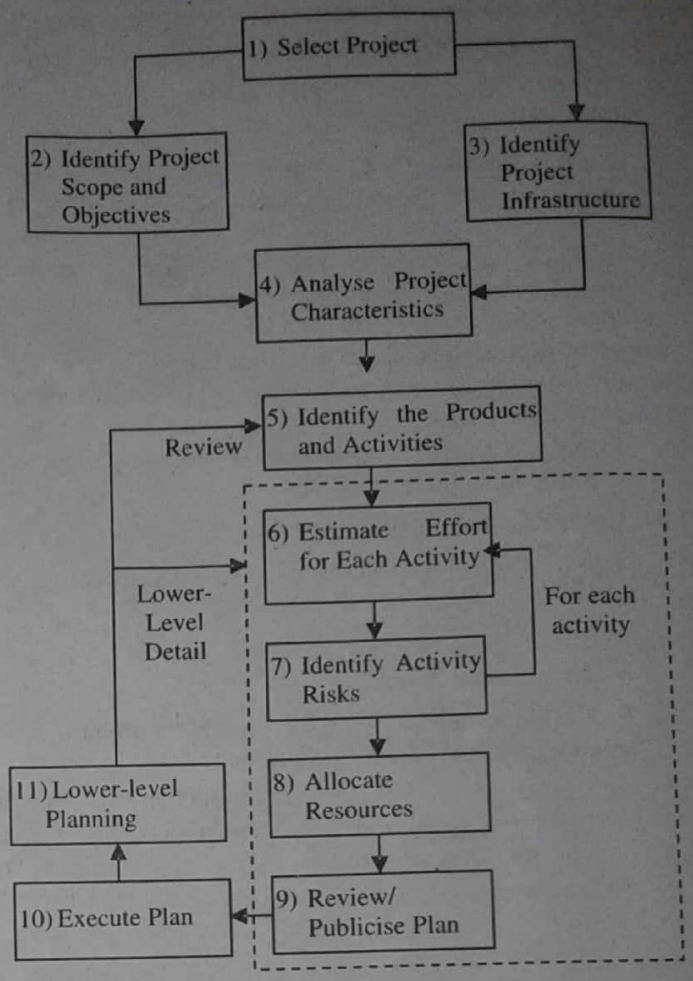


Figure 3.3: Step-Wise Project Planning

2) **Identify Project Scope and Objectives:** This step is comprised of activities which ensure that all the stakeholders of the project agree to its goals and work in a devoted manner so that the project becomes a success. The different activities that make up this stage are:
   Step 1) Identify objectives and practical measures of the effectiveness in meeting those objectives
   Step 2) Establish project authority
   Step 3) Stakeholders analysis
   Step 4) Modify objectives in the light of the stakeholder analysis
   Step 5) establish method of communication

3) **Identify Project Infrastructure:** The project should fit into certain type of infrastructure. If the project manager is new to the organisation, he must get accustomed to the nature of the infrastructure. This step involves the following activities:
   Step 1) Identify relationship between project and strategic planning
   Step 2) Identify installation standards and procedures
   Step 3) Identify project team organisation

4) **Analyse Project Characteristics:** To assure that suitable methods are used to implement the project, this planning operation is undertaken. The important activities that make up this phase are:
   Step 1) Distinguish the project as either objective or product driven

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

Step 2) Analyse other project characteristics (including quality-based ones)

Step 3) Identify high level project risks

Step 4) Take into account user requirement concerning implementation

Step 5) Select development methodology and lifecycle approach

Step 6) Review overall resources estimates

5) **Identify Project Products and Activities:** The specific activities are scheduled in a more detailed manner in this step. The instant steps are planned in much detail while the long-term activities are planned in wide-range. This step comprise of the following major activities:

Step 1) Identify and describes project products (or Deliverables)

Step 2) Document generic product flows

Step 3) Record product instance

Step 4) Produce ideal activity network

Step 5) Modify the ideal to take into account need for stages and checkpoints

6) **Estimate Effort for Each Activity:** In this step the time-span of every activity is calculated after making an evaluation of the essential amount of effort. The work to be done is termed as effort whereas when this figure is distributed by the expected resources is known as duration.

The main activities involved in this step are:

Step 1) **Carry out Bottom-Up Estimates:** Certain estimates of the duration and cost have been already completed in a top-down manner. Estimates of other resources and staff effort are to be made at this point. The elapsed time for every activity can be also produced.

Step 2) **Revise Plan to Create Controllable Activities:** If the activities are assessed individually then it might be revealed that some of the activities are going to take significantly longer amount of time. The project might be challenging to govern if the activities are lengthy.

7) **Identify Activity Risk:** The upcoming undefined events having the possibility of existence and potential for are termed as the risks. Two of the major concerns in any software project are risk identification and management. The following activities are carried out in this step:

Step 1) Identify and quantify activity based risks

Step 2) Plan risk reduction and contingency measures where appropriate

Step 3) Adjust overall plans and estimates to take account of the risks

8) **Allocate Resources:** The method of allocating the existing resources in a cost-effective way is known as resource allocation. According to project management principles, the existing resources are assigned to the activities according to the requirement, keeping in mind the project time. The following activities make up this step:

Step 1) Identify and allocate resources

Step 2) Revise plans and estimates to take into account resource constraints

9) **Review/Publicise Plan:** To confirm the quality and safety of the project a review plan is extremely crucial. The category of actions that are needed at the time of preparing the project plan can be acknowledged using review plans. Various concerns like quality, economic models, engineering methods, etc., are inscribed with the help of the review actions.

Following are the different activities that make up this step:

Step 1) Review quality aspects of the project plan

Step 2) Document plans and obtain agreement

10) & 11) **Execute Plan/Lower Levels of Planning:** After the commencement of the project it is essential to plan every activity in greater detail. The sub-ordinate level and comprehensive planning of the stages should be done at a later time as accessibility of information will increase when the stage begins.

PROJECT ESTIMATION

Ques 7) What is software project estimation?

Or

What are the different steps of software project estimation?

Ans: Software Project Estimation

Software project estimation is the process of estimating various resources required for the completion of a project. Effective software project estimation is an important activity in any software development project. Underestimating software project and under staffing is often leads to low quality deliverables and the project misses the target deadline leading to customer dissatisfaction and loss of credibility to the company.

On the other hand, overstaffing a project without proper control will increase the cost of the project and reduce the competitiveness of the company.

Various project parameters like project size, project duration, effort required to develop, cost etc. need to be estimated. Project estimation is an important planning activity, which aids:

1) Preparing quotations (along with project cost) for customers
2) Resource planning
3) Resource scheduling.

The line of action goes as under (**figure 3.4**).

1) Determine size of the product.

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

2) From the size estimate, determine the effort needed.
3) From the effort estimate, determine project cost

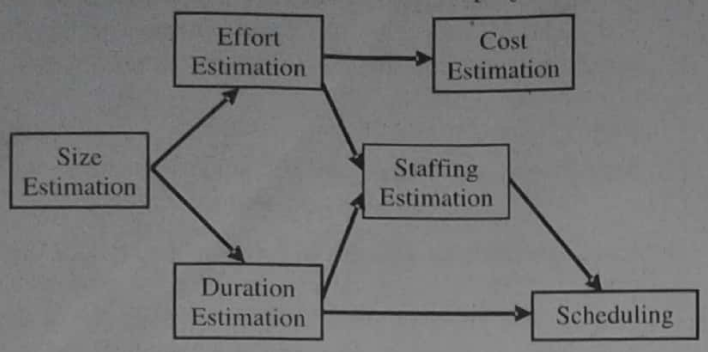


Figure 3.4: Ordering of Estimation Activities

Software Project Estimation Steps

Software project estimation mainly encompasses the following steps:

1) **Estimating the Size of Project:** There are many procedures available for estimating the size of a project, which are based on quantitative approaches like estimating Lines of Code or estimating the functionality requirements of the project called Function point.

2) **Estimating Efforts Based on Person-Month or Person-Hour:** Person-month is an estimate of personal resources required for the project.

3) **Estimating Schedule in Calendar Days/Month/Year Based on Total Person-Month Required and Manpower Allocated to the Project:** Duration in calendar month = Total person-month/Total manpower allocated.

4) **Estimating Total Cost of the Project Depending on the Above and Other Resources:** In a commercial and competitive environment, software project estimation is crucial for managerial decision-making. Project estimation and tracking help to plan and project future projects and provide baseline support for project management and supports decision-making.

Ques 8) Discuss the size estimation in detail.

Or

Discuss about the lines of code and function points metric with example.

Ans: Size Estimation

In order to be able to accurately estimate the project size one need to define some appropriate matric or unit in terms of which we can express the project size. Two metrics under this are as follows:

1) **Lines of Code (LOC):** The lines of code is one of the most popular software sizes metric and has great dependency on the programming language that has been used. The number of lines of code that have been delivered leaving the blank lines and comments is known as LOC.

The programming language that has been selected for the project determines the LOC. When compared to the high-level programming languages like C++ and Java, the lines of code in an assembly language will be much higher. As there is very fewer information during the initial stages of the project, it is necessary to wait till the project ends, to determine the exact number of lines of code.

The protocols that must be used at the time of calculating the LOC has been given below:

i) One logical line of code will result in one line of code.
ii) Test stubs, test drivers, other support software are excluded and only the delivered code that is part of the actual software are considered as lines of code.
iii) Only codes written by the software developer are incorporated while the codes generated by application generators are eliminated.
iv) Lines of code are referred as declarations in the program.
v) Comments are excluded from lines of code.

The project planner estimates three values with the help of the past data. These values are optimistic (S_{opt}), most likely (S_m), and pessimistic (S_{pess}). The following equation can then be used to find an expected value (S):

$$S = (S_{opt} + 4S_m + S_{pess})/6 \qquad .....(1)$$

For example, there are six functions, *viz.*, peripheral control, database management, word processing, word processor, files storage and retrievals, and user interface in software.

With the help of equation (1) the developer has to compute the optimistic, most likely and pessimistic values of every function to estimate the LOC for each function.

For example, the expected size (S) of the user interface function has been calculated and shown in **table 3.1**.

$$S = (2200 + 4 \times 1800 + 1400)/6 = 1800$$

The LOC for the software is 12984 as shown in the **table 3.1**:

Table 3.1: Estimating Size

Function	Pessimistic (S_{pess})	Most Likely (S_m)	Optimistic (S_{opt})	Expected Size (S)
User Interface	1400	1800	2200	1800
Word Processing	1800	2500	3100	2483
File Storage and Retrievals	1700	2200	2500	2167
Database Management	2400	3100	4200	3167
Word Processor	1200	1600	2300	1650
Peripheral Control	1400	1700	2100	1717
Total Estimated Lines of Code(LOC)				12984

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

2) **Function Point:** The functionality delivered by the system is given by the function point metric. With the help of function point, it is possible to estimate the number of components used in the software, forecast the number of errors, calculate the efforts required to design, code, and test the software and determine the number of errors.

An empirical relationship is based on amount of the software complexity and software information domain value is used to derive the function point. Complexity of software can belong to three levels namely simple, average and complex. The combination of all the points that have been listed below is joined to describe the information domain value:

i) **Number of External Inputs (EI):** Distinct application oriented information or data comes from external sources such as users or other applications.

ii) **Number of External Outputs (EO):** The user gets information from the external output produced by the application in the forms of screens, reports, error messages etc. No isolated counting of individual data items on the screen is done.

iii) **Number of External Inquiries (EQ):** The online input that causes the generation of online output immediately is known as external inquiries. Separate counting of each distinct inquiry is done.

iv) **Number of Internal Logical Files (ILF):** Internal logical files are defined as those combinations of data that are part of database or master file and reside within the boundary of the application. External inputs are used to maintain these files.

v) **Number of External Interface Files (EIF):** External interface files are defined as those logical groups of data that are part of data files stored in disk or tape and exist outside the application boundary. The application uses the data granted by the external interface files.

The complexity value for every count can be decided once all the information pertaining to the information domain value has been gathered. This has been shown in **table 3.2**. With the help of the criteria developed using the FP method the organisation is able to conclude which entry is simple, average or complex. For every level of entry a weighting factor is assigned which is in terms of numeric value.

Table 3.2: Computing Function Points

Information Domain Value	Count		Simple	Weighting Factor Average	Complex		
EI		×	3	4	6	=	
EO		×	4	5	7	=	
EQ		×	3	4	7	=	
ILF		×	7	10	15	=	
EIF		×	5	7	10	=	

The following equation estimates the function points in software:

$$FP = \text{Count Total} \times [0.65 + 0.01 \times \sum (f_i)] \ldots (2)$$

Where,

i) **Count total** is the sum of all the FP entries.

ii) **f_i (i = 1 to 14)** are value adjustment factors.

The 14 questions to whose response the value adjustment factors are based are given below:

i) Does the system need reliable recovery and backup?

ii) Does the system need data communication to transfer information?

iii) Does a distributed function exist?

iv) Whether performance is vital or not?

v) Whether the system is running under enormously utilised operational environment?

vi) Does the system need online data entry?

vii) Is it possible to make online data entry (that requires the input transaction) to build over multiple screens or operations?

viii) Does the system permits updating the internal logical files online?

ix) Whether the input, output and inquiries are complex?

x) Whether the internal processing is complex?

xi) Can the code be reused?

xii) Is conversion and installation involved in design?

xiii) Is it possible to install the system at multiple organisations?

xiv) Does the application enables alteration and easy to use?

The questions that have been specified above have been replied in a scale that rates from 0 to 5(**figure 3.5**), where 5 refer to important and 0 refers to not significant. Depending on this information domain the weighted factors in **table 3.2** and the constant values in equation (2) are determined.

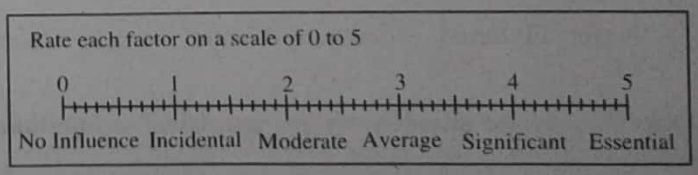


Figure 3.5: Rating of Value Adjustment Factors

Example: It is necessary to calculate the FP counts if estimation of the size is to be made on function points. This can be done using the following equation:

FP Count = Count × Weighting Factor (Average)... (3)

For example, in **table 3.3** FP count for number of user inputs (measurement parameter) is calculated as follows:

FP count = 22 × 4 = 88

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

Table 3.3: Computing Function Points

Measurement Parameter	Count		Simple	Average	Complex		FP Count
Number of User Inputs	22	×	3	4	6	=	88
Number of User Outputs	15	×	4	5	7	=	75
Number of User Inquiries	26	×	3	4	6	=	104
Number Files	6	×	7	10	15	=	60
Number of External Interfaces	2	×	5	7	10	=	14
Count Total							**341**

14 other parameters are considered after the count for each parameter has been determined and total count has been calculated. This has been shown in **table 3.4**.

Table 3.4: Value Adjustment Factors

Factors	Value
Backup and recovery	3
Data communications	1
Distributed processing	1
Performance critical	3
Existing operating environment	2
Online data entry	3
Input transactions over multiple screens	4
Master files updated online	2
Information domain value complex	4
Internal processing complex	4
Code design for reuse	3
Conversions/installation in design	2
Multiple installations	4
Application design for change	4
Value Adjustment Factor	40

The function point is calculated using equation (2):

$$FP = 341 \times [0.65 + (0.01 \times 40)] = 358$$

Note: FP-based estimation is more complex than LOC.

Ques 9) Write short notes on empirical estimation models.

Ans: Empirical Estimation Models

An estimation model for computer software uses empirically derived formulas to predict effort as a function of LOC or FP. The empirical estimation model uses empirically (experimentally) derived formulae to predict effort as a function of lines of code or function point. In empirical estimation model, the empirical values of LOC or FP are put. The empirical model supports a limited number of software projects. Typical estimation model is derived using regression analysis on data collected from the past software projects.

Some **examples** of empirical models are like:

1) Putnam's Model
2) COCOMO (COnstructive COst MOdel)

The **general form of empirical estimation models** are shown below:

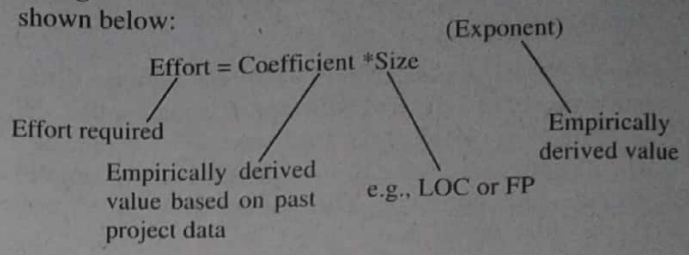


Figure 3.6: General Form of Empirical Estimation Models

Among the many LOC-oriented estimation models proposed, some are as follows:

$E = 5.2 \times (KLOC)^{0.91}$ (Walston-Felix Model)

$E = 5.5 + 0.73 \times (KLOC)^{1.16}$ (Bailey-Basili Model)

$E = 3.2 \times (KLOC)^{1.05}$ (Boehm Simple Model)

$E = 5.288 \times (KLOC)^{1.047}$ (Doty Model for KLOC > 9)

FP-oriented models have also been proposed. These include:

$E = -13.39 + 0.0545\,FP$ (Albrecht and Gaffney Model)

$E = 60.62 \times 7.728 \times 10^{-8}\,FP^3$ (Kemerer Model)

$E = 585.7 + 15.12\,FP$ (Matson, Barnett, and Mellichamp Model)

Ques 10) What is COCOMO? What are the different stages in software cost estimation by COCOMO?

Or

What are the drawbacks of intermediate and basic COCOMO models?

Or

What is basic, intermediate and complete COCOMO model?

Ans: Constructive Cost Model (COCOMO)

The **COCOMO** (**CO**nstructive **CO**st **MO**del) was developed by **Barry Boehm** in the early 80's with an intention of estimating the total effort requirement in a software project development. As the model has its basis in the previously established projects, it is used most frequently. During the total effort estimation management, cost of development and other support tasks including the cost of secretarial and other staff are also taken into concern. The size is measured in terms of KDLOC (thousands of Delivered Lines of Code) in this model.

Stages in Software Cost Estimation by COCOMO

Three stages are there to estimate software cost:

1) **Basic COCOMO Model:** An approximate estimation is given by the basic COCOMO model:

$$Effort = a_1 (KLOC)^{a_2}\ PM$$

$$T_{dev} = b_1 (Effort)^{b_2}\ Months$$

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

Where,

i) KLOC is the estimated kilo lines of source code,

ii) a_1, a_2, b_1, b_2 are constants for different categories of software products,

iii) T_{dev} is the estimated time to develop the software in months,

iv) Effort estimation is obtained in terms of Person Months (PMs).

Development Effort Estimation

For calculating the effort based on size for the three types of systems are given:

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

ii) **Semi-Detached:** Effort = $3.0\ (KLOC)^{1.12}$ PM

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

Development Time Estimation

For the estimation of time based on the effort following are the formulas for the three classes of software:

i) **Organic:** $T_{dev} = 2.5\ (Effort)^{0.38}$ Months

ii) **Semi-detached:** $T_{dev} = 2.5\ (Effort)^{0.35}$ Months

iii) **Embedded:** $T_{dev} = 2.5\ (Effort)^{0.32}$ Months

For example, an organic software product has the size of 32,000 lines of source code:

Then, Effort = $24^* (32)^{1.05} = 91$ PM

Nominal development time = $2.5^*(91)^{0.38} = 14$ months

2) **Intermediate COCOMO Model:** According to the basic COCOMO model, the development time and effort are considered to be functions of only the product size.

Though, there are other parameters associated with the project that have significant effect on the product development effort and time to develop the product.

Thus, it is necessary to take into account the effect of all the associated parameters so that the estimation of the project duration and effort required can be done accurately.

This fact has been addressed by the intermediate COCOMO model and the refinement of the primary estimate that is attained from the basic COCOMO model expression is done with the help of 15 cost drivers or multipliers that have been derived from several software development attributes.

According to Boehm, the project manager has to rate all the 15 parameters for a project on a one to three scale. Once done, he recommends relevant cost drivers for the basic COCOMO model generated from primary estimate.

This model is said to be a refinement of the basic COCOMO model. In this model, the computation of the software cost is done using a set of "cost driver attributes". The four cost driver attributes are given below:

Table 3.5: COCOMO Intermediate Cost Drivers

Driver Type	Code	Cost Driver
Product Attributes	RELY	Required software reliability
	DATA	Database size
	CPLX	Product complexity
Computer Attributes	TIME	Execution time constraints
	STOR	Main storage constraints
	VIRT	Virtual machine volatility – degree to which the operating system changes
	TURN	Computer turnaround time
Personnel Attributes	ACAP	Analyst capability
	AEXP	Application experience
	PCAP	Programmer capability
	VEXP	Virtual machine(i.e., operating system) experience
	LEXP	Programming-language experience
Project Attributes	MODP	Use of modern programming practices
	TOOL	Use of software tools.
	SCED	Required development schedule

Additional attributes that have effect on software development except those mentioned earlier are:

i) Programming language, and

ii) Estimated lines of code (unadjusted and adjusted for the use of existing software).

Table 3.6 shows numerical rating after mapping of the categorical values to the successive attributes (VL, LO, NM, etc.):

Table 3.6: Intermediate COCOMO Attributes

	VL	LO	NM	HI	VH	XH
RELY	0.75	0.88	1.00	1.15	1.40	
DATA		0.94	1.00	1.08	1.16	
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
TIME			1.00	1.11	1.30	1.66
STOR			1.00	1.06	1.21	1.56
VIRT		0.87	1.00	1.15	1.30	
TURN		0.87	1.00	1.07	1.15	
ACAP	1.46	1.19	1.00	0.86	0.71	
AEXP	1.29	1.13	1.00	0.91	0.82	
PCAP	1.42	1.17	1.00	0.86	0.70	
LEXP	1.14	1.07	1.00	0.95		
VEXP	1.21	1.10	1.00	0.90		
MODP	1.24	1.10	1.00	0.91	0.82	
TOOL	1.24	1.10	1.00	0.91	0.83	
SCED	1.23	1.08	1.00	1.04	1.10	

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

The final product multiplier (P) is obtained by multiplying the partial results after rating each attribute depending on the product. The following is the expression of the effort formula:

$$\textbf{Effort} = \textbf{Effort}_{nom} * P$$

Where, $Effort_{nom}$ arises in the following form depending on the type of software:

For embedded systems:

$$Effort_{nom} = 2.8 * KDLOC^{1.20}$$

For organic systems:

$$Effort_{nom} = 2.8 * KDLOC^{1.05}$$

For semi-detached systems:

$$Effort_{nom} = 2.8 * KDLOC^{1.12}$$

The relationship given here is the same as the basic model in which b remains same and only 'a' has been changed. The data that has been used in the estimation procedure are values of the parameters of the effort model that have been constructed using regression models. In case, an attribute has insignificant rating then the values of effort remains unchanged, and it has no influence on the values of P.

Some of the attributes might have to compensate, i.e., if values in one attribute rises then there is a reduction in the other attributes. The development schedule estimate is equivalent to the basic model. To calculate the support effort, the following formula is used:

$$\textbf{Effort}_{maintenance} = \textbf{ACT} * \textbf{Effort}_{nom} * P$$

According to Boehm, sound results are created by the intermediary COCOMO model "within 20%-68% of the time" (this has been tested in the COCOMO database).

3) **Complete COCOMO Model:** The **drawbacks of intermediate and basic COCOMO models** have been given below:
   i) The software product is considered to be a single homogeneous entity.
   ii) Most of the systems are made up of many small sub-systems. These sub-systems come under various software classes such as embedded, organic, etc. Even the reliability requirements may vary for each sub-system.
   iii) Separate estimation of cost for each sub-system is made.
   iv) The total cost is attained by adding the individual cost of all the sub-systems.
   v) The error or margin is decreased in the final estimate.

For example, in the following project the complete COCOMO model has been applied. An organisation has various offices located at different places of the country. Then a distributed Management Information System (MIS) developed for the organisation might have the following sub-systems:

i) Database part,
ii) Graphical User Interface (GUI) part, and
iii) Communication part.

Among these the communication component comes under the embedded software class, while the GUI part belongs to organic type software and database could be semi-detached type. To get the overall cost of the system it is necessary to sum up the cost of the three components after they have been separately estimated. The historical project database of the organisation can be used to fine-tune and validate the various parameter values of the model to enhance the exactness of the consequences. COCOMO model is devoid of full scientific justification and is not entirely accurate. However, the COCOMO model is useful in providing an engineering approach to the management of software projects.

Ques 11) Write short note on static, single and multivariable variable models.

Or

Discuss the resource models in detail

Ans: Resource Models

Basili(1980) described a class of resource models. Some of them are as follows:

1) **Static, Single Variable Models:** Methods using this model use an equation to estimate the desired values such as cost, time, effort, etc. They all depend on the same variable used as predictor (say, size). An **example** of the most common equation is:

$$C = a L^b \quad .....(1)$$

Where C is the cost (effort expressed in any unit of manpower, e.g., person-months) and L is the size generally given in the number of lines of code. The constants, a and b are derived from the historical data of the organisation. Since a and b depend on the local development environment, these models are not transportable to different organisations.

The Software Engineering Laboratory of the University of Maryland has established a model, the SEL model, for estimating its own software productions. This model (BASL80) is a typical example of a static single-variable model.

$$E = 1.4 L^{0.93} \quad .....(2)$$

$$DOC = 30.4 L^{0.90} \quad .....(3)$$

$$D = 4.6 L^{0.26} \quad .....(4)$$

Effort (E in Person-months), documentation (DOC, in number of pages) and duration (D, in months) are calculated from the number of lines of code (L, in thousands of lines) used as a predictor.

COCOMO is an **example** of a static single-variable model.

2) **Static, Multivariable Models:** Although these models are often based on equation (4.1), they actually depend on several variables representing various aspects of the software development environment, e.g., methods used, user participation, customer-oriented changes, memory constraints, etc. The model developed by **Walton and Felix** at IBM

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

provides a relationship between delivered lines of source code (L in thousands of lines) and effort E (E in person- months) and is given by the following equation:

$$E = 5.2\,L^{0.91} \qquad .....(5)$$

In the same fashion, the duration of the development (D in months) in given by:

$$D = 4.1L^{0.36} \qquad .....(6)$$

Data collected on 60 software projects, representing a wide variety of applications and size (ranging from 4,000 to 4,67,000 lines of code), shows a relationship between productivity (expressed in number of lines of source code per person months) and a productivity index I.

The productivity index uses 29 variables which are found to be highly correlated to productivity as follows:

$$I = \sum_{i=1}^{29} W_i X_i \qquad .....(7)$$

Where W_i is a factor weight for the ith variable and X_i = [–1, 0, + 1]. The estimator gives X_i one of the values –1, 0 or +1 depending on whether the variable decreases, has no effect, or increases the productivity respectively. The terms of equation (7) are then added up to give the productivity index.

Ques 12) Discuss the Staffing and Personal Planning in detail.

Or

What factors are to be considered during staff selection?

Ans: Staffing and Personal Planning

Once the cost and the overall duration of the project, the average staff size for the project can be determined by dividing the total effort (in person-months) by the overall project duration (in months).

Average Staff Size = total effort (in person-months)/ overall project duration (in months)

The average staff size is not detailed enough for proper personnel planning, especially if the variation between the actual staff requirement at different phases is large. Typically the staff requirement for a project is small during the requirement and design, and is the maximum during the implementation and testing, and then again drops during the final phases of integration and testing.

For personnel planning and scheduling, it is useful to have effort and schedule estimates for the sub-systems and basic modules in the system. At the planning time, when the system design has not been done, the planner can only expect to know about the major subsystems in the system, and perhaps the major modules in these subsystems.

Factors to be Considered During Staff Selection

If anyone has some choice of staff, the factors that may influence decision are **given below**. The most important factors vary depending on the application domain, the type of project and the skills of other members of the project team:

1) **Application Domain Experience:** For a project to develop a successful system, the developers must understand the application domain. It is essential that some members of a development team have some domain experience.

2) **Platform Experience:** This may be significant if low-level programming is involved. Otherwise, this is not usually a critical attribute.

3) **Programming Language Experience:** This is normally only significant for short-duration projects where there is not enough time to learn a new language. While learning a language itself is not difficult, it takes several months to become proficient in using the associated libraries and components.

4) **Problem Solving Ability:** This is very important for software engineers who constantly have to solve technical problems. However, it is almost impossible to judge without knowing the work of the potential team member.

5) **Educational Background:** This may provide an indicator of what the candidate knows and his or her ability to learn. This factor becomes increasingly irrelevant as engineers gain experience across a range of projects.

6) **Communication Ability:** Project staff must be able to communicate orally and in writing with other engineers, managers and customers.

7) **Adaptability:** Adaptability may be judged by looking at the experience that candidates have had. This is an important attribute, as it indicates an ability to learn. Attitude Project staff should have a positive attitude toward their work and should be willing to learn new skills. This is an important attribute but often very difficult to assess.

8) **Personality:** This is an important attribute but difficult to assess. Candidates must be reasonably compatible with other team members. No particular type of personality is more or less suited to software engineering.

Ques 13) What do you understand by Rayleigh Curve?

Ans: Norden's /Rayleigh Curve

Norden thought that number of personnel required during any development project is not constant.

Norden analysed many R&D projects, and observed Rayleigh curve represents the number of full-time personnel required at any time.

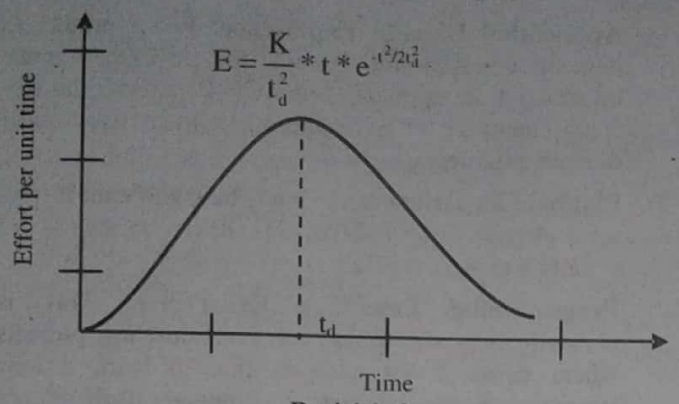
Norden found that the staffing pattern can be approximated by the Rayleigh distribution curve. He represented the Rayleigh curve by the following equation:

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

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

Where E is the effort required at time t. E is an indication of the number of engineers (or the staffing level) at any particular time during the duration of the project.



Rayleigh Curve

K is the area under the curve, and t_d is the time at which the curve attains its maximum value.

It must be remembered that the results of Norden are applicable to general R & D projects and were not meant to model the staffing pattern of software development projects.

Ques 14) Explain the following model:
1) **Putnam's Resource Allocation Model**
2) **Jensen's Model**

Ans: Putnam's Resource Allocation Model

Putnam studied the problem of staffing of software projects and found that software development has characteristics very similar to any other R & D projects studied by Norden and that the Rayleigh-Norden curve can be used to relate the number of delivered lines of code to the effort and the time required to develop the product. By analysing a large number of army projects, Putnam derived the following expression:

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

Where,
i) K is the total effort expended (in PM) in the product development
ii) L is the product size in KLOC.
iii) t_d corresponds to the time of system integration and testing. Therefore, t_d can be approximately considered as the time required to develop the software.
iv) C_k is the state of technology constant and reflects constraints that impede the progress of the programmer.

The value of $C_k = 2$ corresponds to poor development environment (no methodology, poor documentation, and review, etc.), $C_k = 8$ corresponds to good software development environment (software engineering principles are adhered to), and $C_k = 11$ corresponds to an excellent environment (in addition to following software engineering principles, automated tools and techniques are used). The exact value of C_k for a specific project can be computed from the historical data of the organization developing it.

Putnam suggested that optimal staff build-up on a project should follow the Rayleigh curve. Only a small number of engineers are needed at the beginning of a project to carry out planning and specification tasks. As the project progresses and more detailed work is required, the number of engineers reaches a peak. After implementation and unit testing, the number of project members fall.

Thus, 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 a constant number of engineers are used over all the phases of a project, some phases would be overstaffed and some understaffed leading to inefficient use of manpower, and consequent schedule slippage and increase in cost.

If we examine the Rayleigh curve, we can see that approximately 40% of the area under the Rayleigh curve is to the left of t_d and 60% to the right of t_d. This has been verified mathematically by integrating the expression provided by Putnam. This implies that the ratio of the effort required to develop the product to its maintenance effort is approximately 40:60. The Putnam estimation model works reasonably well for very large system but seriously over estimates the effort on medium and small systems.

Jensen's Model

Jensen model is very similar to Putnam model. However, it attempts to soften the effect of schedule compression on effort to make it applicable to smaller and medium sized projects. Jensen proposed the equation:

$$L = C_{te} t_d K^{1/2}$$

Or
$$\frac{K_1}{K_2} = \frac{(t_{d_2})^2}{(t_{d_1})^2}$$

Where C_{te} is the effective technology constant, t_d is the time to develop the software, and K is the effort needed to develop the software. When the project duration is compressed, Jensen model gives the increase in effort (and cost) requirement proportional to the square of the degree of compression.

SOFTWARE DESIGN

Ques 15) What do you understand by software design phase?

Or

Define the term software design. What are the different types of design?

Or

What is design process?

Ans: Software Design Phase

During the Design Phase, the system is designed to satisfy the requirements identified in the previous phases. The requirements identified in the Requirements Analysis Phase are transformed into a System Design Document that accurately describes the design of the system and that can be used as an input to system development in the next phase.

The purpose of the design phase is to transform the requirements into complete and detailed system design

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

specifications. Once the design is approved, the Development Team begins the Development Phase.

The software design phase begins after the formulation of the requirements document. This deals with the solution to the problem mentioned in requirements specification stage.

During the design phase, customer and business requirements as well as technical considerations all gather to coordinate for a new product or system.

The design process has a set of principles, concepts, and practices which help the software engineer to build the new product or system. This new model of the product or system is put to quality test before review. After this, code is generated and put to further tests.

The design model mentions details pertaining to software data structures; architecture, interfaces and components necessary for implementation of the system. Design is a representation of the customer's requirements and simultaneously of the quality standard of "good" design of an organisation.

For achievement of well-engineered software, software design is a prerequisite and is validated by software engineering practitioners across the globe. Software design is a creative process as it deals with the development of mechanics of the new system.

According to Coad and Yourdon, "Software Design is the practice of taking a specification of externally observable behaviour and adding details needed for actual computer system implementation, including human interaction, task management, and data management details".

According to Stevens, "Software Design is the process of inventing and selecting programs that meet the objectives for a software system".

According to Association of Software Design (ASD), "Software design sits at the crossroads of all the computer disciplines: hardware and software engineering, programming, human factors research, ergonomics. It is the study of the intersection of human, machine, and the various interfaces – physical, sensory, psychological – that connect them".

A Software Design Document (SDD) results from requirements determination. The main purpose of the design phase is to provide a solution to a problem mentioned in SRS.

Figure 3.7 shows a framework of design. It starts with initial requirements and ends up with the final design.

Here first data is gathered on user requirement and analysed accordingly. After analysis a high-level design will be prepared and design will be validate against requirements on regular basis and design will be refined in every cycle.

Finally it is documented to produce software design document.

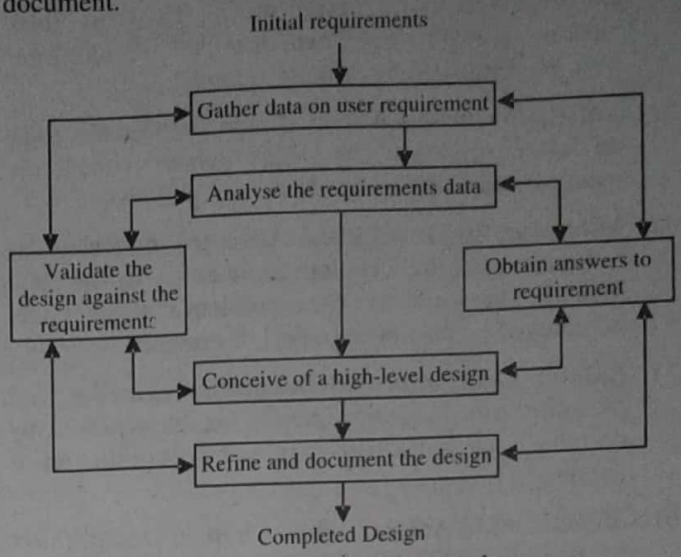


Figure 3.7: Design Framework

Types of Design

1) **Preliminary (or High-Level or Architectural) Design:** By high-level design we would mean identification of different modules, the control relationships and the definitions of the interfaces among them. The outcome of high-level design is called the program structure or software architecture. A popular way is to use a tree-like diagram caned the structure chart to represent the control hierarchy in a high-level design.

2) **Detailed Design:** During detailed design, the data structure and the algorithms of different modules are designed. The outcome of the detailed design stage is usually known as the module specification document.

Design Process

Software design is generally considered a two-step process:

1) Architectural design (also referred to as high-level design and top-level design) describes how software is organized into components.

2) Detailed design describes the desired behavior of these components.

The output of these two processes is a set of models and artifacts that record the major decisions that have been taken, along with an explanation of the rationale for each nontrivial decision. By recording the rationale, long-term maintainability of the software product is enhanced.

Ques 16) What are the different design principles?

Ans: Design Principles

Davis has propounded the following principles of software design:

1) **Process should not suffer from "Tunnel Vision":** A good designer must consider all possible approaches without being judgmental about any particular one. Each alternative should be judged based on the requirements of the problem, available resources, and the different design models.

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

2) **Traceable to Analysis Model:** Multiple requirements are catered to by the design model. Therefore, there must be ways to track whether such requirements have been fulfilled by the design model.

3) **Not Re-invent the Wheel:** Design time should come up with original new ideas which should be incorporated with previously present patterns.

4) **Minimise the Intellectual Distance between the Software and the Problem as it exists in the Real World:** The structure of the problem domain should be imitated by the software design structure.

5) **Exhibit Uniformity and Integration:** There should be uniformity in design which can be achieved by defining style rules and formats before initiation of the design process.

6) **Structured to Accommodate Change:** Designs must be flexible and extensible so that they can comply with changes.

7) **Structured to Degrade Gently even when Unusual Circumstances are encountered:** Software must be designed in a manner that it terminates processing gracefully when required.

8) **Design is not coding, Coding is not design.**

9) **Assessed for Quality as it is being created:** A selection of design concepts and measures are used to assess quality of design while the process is on.

10) **Reviewed to Minimise Conceptual Errors:** Omissions, ambiguities and inconsistencies must be addressed by the design team keeping aside syntax of the design model.

Ques 17) What are the main objectives of software design?

Ans: Objectives of Software Design

Although the goodness of a design is dependent on the application being developed but there are certain objectives that all good design must possess:

1) **Correctness:** A good design must encompass all requirements of the system i.e. all explicit requirements mentioned in the analysis model and implicit requirements mentioned by the customer.

2) **Understand ability:** Design must be easy to read, simple and easy to understand, so that those who are responsible for generating code as well as those who will test the code can both easily understand. One way by which this can be achieved is by assigning meaningful names to each design component.

3) **Modularity:** One of the basic characteristic of good designing is modularity i.e. the problem to which solution is to be found is broken down into modules which are designed independent of each other. This leads to reduction of complexity of design solution thereby helping in precise arrangement of modules.

4) **Completeness:** Different components of a design should be clearly mentioned for a good design. The important components of relevant data structures, modules' external interfaces and module interconnections should be encompassed.

5) **Efficiency:** Another hallmark of good design is its efficiency which means it should engage in optimal use of its resources like **processor time** and **memory**. An efficient system should use less of these two resources.

6) **Consistency:** There should be no inconsistency in design. It should be uniform throughout the system.

7) **Verifiability:** Designs are confirmed for accuracy, as they cannot afford to be incorrect. Verifiability measures how easily a design can be checked. It should be such that verification techniques can be easily applied to design.

8) **Traceability:** Designs should be traceable as it helps in verification of design. Design elements should be noticeable to the requirements specification.

Ques 18) Discuss the basic concepts of software design.

Or

Write short notes on the following:

1) **Abstraction**
2) **Modularity**
3) **Information Hiding**
4) **Functional Independence**
5) **Control Hierarchy**
6) **Problem Partitioning**

Ans: Basic Concepts of Software Design

Some concepts related to software design are as follows:

1) **Abstraction:** Abstraction is one of the most important tools used in all engineering disciplines. This allows the designer to consider a component at an abstract level without thinking about the implementation of the component.

   An abstraction of a component defines the external behaviour of the component. This does not include the internal reasons which produced such behaviour. As such, the abstract meaning of a component is simpler than the component.

2) **Modularity:** Well-designed software systems are characterised by modularity. A system can be said to be modular if it has independent, simple and well-defined units or **modules** which interact through interfaces.

   A module is an independent part of a system and can be separated from program which can be acknowledged by compiling and loading. In common programming language, a module could be a function, procedure, process or a package. A module is typically a procedure of function or a group of both in systems using functional abstraction.

3) **Information Hiding:** The fundamental design concept for software is information hiding. Using this principle, each module in the system hides internal

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

details of its processing activities wherein modules only communicate through well-defined interfaces.

Effective modularity can be achieved when modules (independent) communicate with one another with only necessary information required for software function. Information hiding is necessary because:

i) Reduces 'side effects",
ii) Reduces the global impact of local design decisions,
iii) Stresses on communication via controlled interfaces,
iv) Does not encourage global data use,
v) It provides encapsulation which is a necessary attribute of high quality design, and
vi) Produces better quality software.

Information hiding is the main design theory which is used for modularisation standard in combination with other design methods or as an architectural design of a system.

4) **Functional Independence:** Functional independence results from modularity, abstraction, and information hiding. Functional independence results from developing modules with single function, without excessive interaction with other modules. An independent module has very less interaction with other modules.

Independence is assessed using two qualitative criteria: **cohesion** and **coupling.**

i) **Cohesion** is an indication of the relative functional strength of a module.
ii) **Coupling** is an indication of the relative interdependence among modules.

5) **Control Hierarchy:** Control hierarchy is also referred to as **program structure** which represents how the program components are organised and shows the hierarchy of control. It is not applicable to all architectural styles neither does it characterise procedural aspects of software like process sequence, order of decisions or operations repetition. The control relationship between modules-super-ordinate and sub-ordinate modules is expressed.

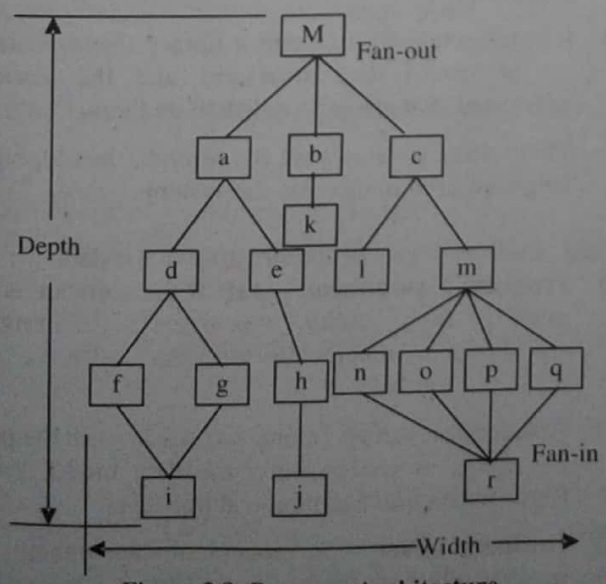


Figure 3.8: Program Architecture

Figure 3.8 shows the hierarchical control of the call and return architectures:

The number of levels and overall extent of control are specified by the **depth** and **width** as shown in **figure 3.8**.

Fan-out shows the number of modules which are straightly controlled by other modules whereas **Fan-in** shows the number of modules which control a particular module.

6) **Problem Partitioning:** Large complex problems can be solved faster and more accurately by the "divide and conquer" principle. A smaller problem can be tackled and resolved easily than larger problems.

When problems are divided into small pieces, then each can be solved separately. This is the main logic behind this principle. Since each part needs to be solved separately, dividing a larger problem into smaller pieces becomes a complex task which many methodologies for system design aim to address. Problem partitioning may lead to hierarchies in design i.e. hierarchy of components. The method is used to define the inter-relationships among the different components.

Types of Problem Partitioning

Various types of problem partitioning are as follows:

i) **Horizontal Partitioning:** In this, there are separate branches of module hierarchy for each program function. **Figure 3.9** depicts horizontal partitioning with control modules in dark shade which enable co-ordination of communication between an implementation of functions.

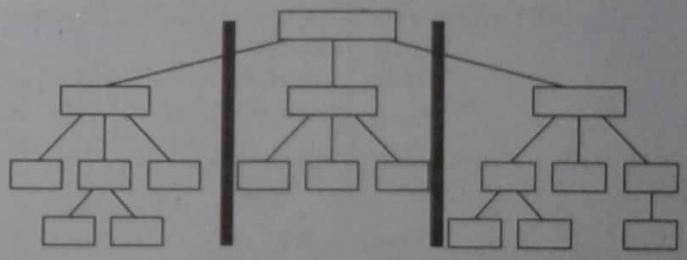


Figure 3.9: Horizontal Partitioning

Horizontal partitioning has three partitions according to the simplest approach – input, data transformation (processing) and output.

ii) **Vertical Partitioning:** This is also referred to as factoring where control and work are distributed top-down within the program structure. Top level modules must have the controls with little processing work. On the other hand, lower level modules are the workers who perform the input, computation and output jobs.

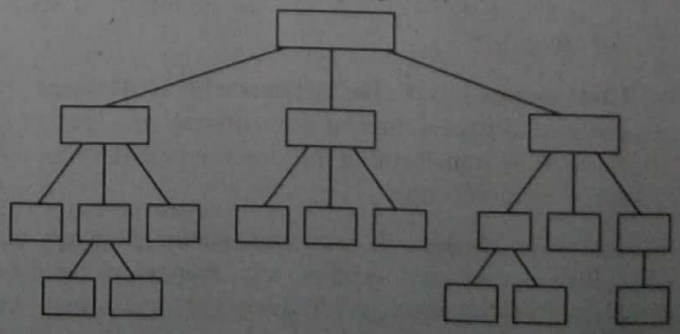


Figure 3.10: Vertical Partitioning

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

Figure 3.10 shows a vertical partitioning wherein a change in control module has a cascading effect on the sub-ordinate modules. But a change in the lower level module like the worker will produce very less side effects. Vertical partitioning ensures a stronger and more maintainable structure as it produces less side effects even when changes are made to the system.

Ques 19) What are the different design methodologies?

Or

What are the different elements of design model?

Ans: Design Methodologies/Classification of Software Design

There are four elements that are used to develop the complete specification of the design model (**figure 3.11**). They have been discussed in detail below:

1) **Data Design:** The data objects that are identified in the analysis phase are used to create the data structure. The data design activity is based on the data objects, their attributes and the relationship between them as provided by the entity relationship diagrams. As the data design lays the foundation of the various other design elements, it serves as the first step of the design engineering process.

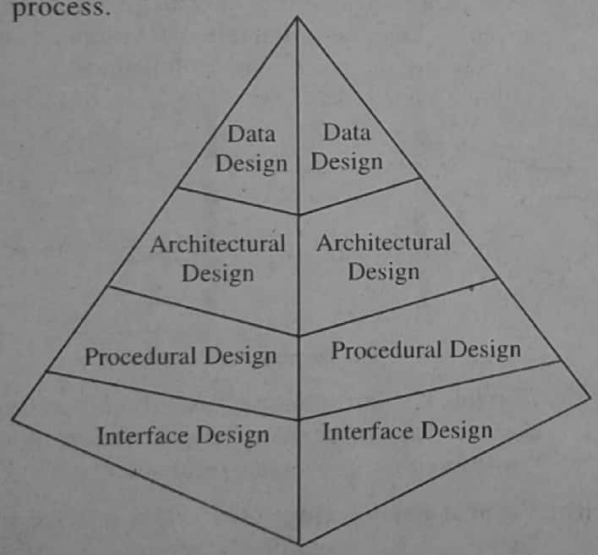


Figure 3.11: Design Model and its Elements

2) **Architectural Design:** The relationship between the various aspects that affects the implementation of the architecture such as design patterns, structural elements of the software and the architectural styles are identified.

3) **Component-Level Design/Procedural Design:** A procedural explanation of the software constituents is attained by transforming the structural elements of the software architecture.

4) **Interface Design:** The communication between the software and the system, and the inter-operation among themselves, and also the end-users are described in interface design.

Ques 20) What is data design?

Or

What are the main principles of data design?

Or

What are the three levels of the structure of the data?

Ans: Data Design

The most significant design activity that must be undertaken at the beginning is **data design**. Selecting the suitable data structure is the key concern which is now shifting the focus of the data design to the data structures.

Efficient, modular and less complex program structure can be obtained by making the data design the first design activity. Data structures which are necessary for the software implementation are obtained by converting the information domain model of the analysis phase. The data design activity gets its base from relationships, data objects and attributes shown in entity relationship diagrams and information provided by the data dictionary.

The specification of the data types along with their integrity rules are identified through the data design process. Certain principles must be followed so that efficient data structures can be identified and designed.

The main **principles** are listed below:

1) It is necessary to identify the data structures needed to implement the software along with the operations to be performed on them.

2) In order to show the interaction between the various data objects and the constraints to be imposed on them, a data dictionary must be developed.

3) The design decisions should be taken in detail in the later phases and the data design process must involve stepwise refinement.

4) The representation of the data structure should be only conveyed to those modules that need to directly access the data that the data structure contains.

5) It is necessary to maintain a library that contains the set of useful data structures and the associated operations that are accomplished on them.

6) There must be a support for abstract data type in the language used to develop the system.

There are three levels of the structure of the data:

1) **Program Component Level:** If the software is to be made of high quality, it is essential to design the algorithms to manipulate the data structures at this level.

2) **Application Level:** In order to accomplish the precise objectives of the business the data model must be transformed into a database at this level.

3) **Business Level:** The process of data mining has a notable effect on the business. Thus it is necessary to re-organise the information stored in various databases into data warehouse at this level.

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

Ques 21) What is architectural design?

Or

What do you understand by high level design?

Ans: Architectural Design

The architecture of the software identifying its components and defining the top level structure of the software is obtained by converting its necessities. The architectural design or system design acts as the 'blueprint' that facilitates the development of the software.

The relationship among the major structural elements is defined by the architectural design. The representation of the control relationship between the modules and development of a modular structure is the major goal.

The structure of program components and data structure to build the computer system is represented by the architectural design which is also known as the **high level design**.

Multiple representations of the architectural structure of the system are derived by starting the architectural design phase with the data design. The structure that best suites the quality attributes and customer requirements are derived by analysing the alternate architectural patterns or styles. An architectural design method is used to elaborate the architecture after the selection of an alternative has been done.

During the architectural design, an architectural model that encompasses the program structure and data architecture is created.

According to IEEE, "Architectural design is the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system."

Following are the various sources of architectural design:

1) Information related to the domain of the application for which software development is to be carried out.
2) Use of data flow diagrams.
3) Making architectural styles and architectural patterns accessible.

Ques 22) What are the main functions of architectural design?

Ans: Functions of Architectural Design

The following functions are performed by an architectural design:

1) An abstraction level at which the performance and functional behaviour of the system can be defined by the architectural design.
2) Those features of the system that can be effortlessly altered without having any effect on the integrity of the system are defined by the guidelines provided by the architectural design.
3) All top-level designs are estimated by it.
4) The top-level design for the internal and external interfaces are developed and documented by it.
5) The preliminary versions of the user documentation are also developed by it.
6) The schedule for software integration is defined and the preliminary test requirements are documented.

Ques 23) What models are used for representing architectural design?

Ans: Model for Representing Architectural Design

The following models represent the architectural design:

1) **Structural Model:** The architecture is demonstrated as a collection of program components in a systematic manner.
2) **Dynamic Model:** The behavioural aspect of the software architecture is specified. An indication of the change of the system or structure configuration is also given with changes in function due to external environment changes.
3) **Process Model:** The focus is on the technical or business processes that have to be implemented in the system.
4) **Functional Model:** The functional hierarchy of the system is represented by it.
5) **Framework Model:** Architectural design patterns that have been encountered in analogous kinds of applications are acknowledged which raises the level of abstraction.

Ques 24) What are the different architectural styles?

Or

Explain the following:

1) **Data-Flow Architecture**
2) **Object-Oriented Architecture**
3) **Layered System Architecture**
4) **Data-Centred Architecture**
5) **Call and Return Architecture**

Ans: Architectural Styles

A group of systems that are inter-linked and share semantic and structural properties are defined by the architectural styles. Thus establishing a structure for all the components making up the system is the aim of the architectural styles. Architectural styles that are generally used are:

1) **Data-Flow Architecture:** This architecture is implemented on those systems that apply numerous transformations to generate the desired output after accepting the input. The components are known as filters which transform the data. The connectors known as **pipes** are then used to send the transformed data to other filters for the next phase of processing. The **filters** are not concerned with the producer or consumer of data and works as an independent entity.

   The channel provided by pipe is unidirectional in nature and transports data received at one end to the other end. The data is only provided to the filter at the receiver end and there is no change in data.

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

A batch sequential system is dis-integrated by the data-flow architecture frequently. This system applies a series of sequential filters for transforming data after accepting a batch of data. A UNIX shell program is a mutual illustration of this architecture. The UNIX processes perform the function of filters and the file system through which the interaction takes place performs the function of pipes in such programs. Some other popular illustrations of this architecture are distributed systems, parallel programming, compilers, functional programming and signal processing systems.

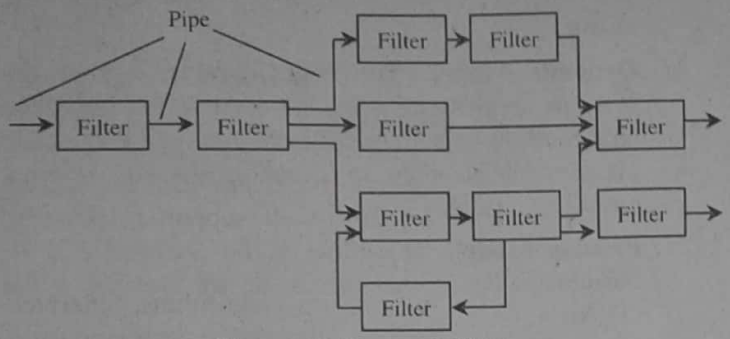


Figure 3.12: Pipes and Filters

2) **Object-Oriented Architecture:** The data and operations that are used to manipulate the data are encapsulated by the components of the system in this style. The components are termed as objects and communication occurs between them via methods or connectors.

The two main **characteristics** of this architectural style have been given below:

i) The integrity of the system is sustained by the objects, and

ii) One object is unaware of how the other objects are represented.

The **advantages** associated with the object-oriented architecture have been listed below:

i) With this style, the designer is able to obtain a collection of independent objects from the problem.

ii) It is possible to change one object without affecting the other objects, as their implementation details are veiled.

3) **Layered System Architecture:** This architecture is designed with an aim of minimising the interaction between the layers by creating a hierarchy of client-server processes. In layered architecture, each layer acts as a server to the layer below it and client to the layer above it.

Multiple layers or components are defined in layer architecture and each layer performs a set of tasks that are well defined. Each layer is built on the layer below it and arrangement is done in the hierarchical manner. A layer provides numerous services to the layer below it and acts as a client to the layer above it. There are protocols or connectors that define the set of rules to be followed at the time of interaction between the layers. The OSI-ISO (Open Systems Interconnection-International Organisation for Standardisation) communication system is an example of this architecture.

Application is defined by the higher levels while hardware is described by the lower levels, in these systems. The designs of the layered system are supported as there is an increase in the level of abstraction.

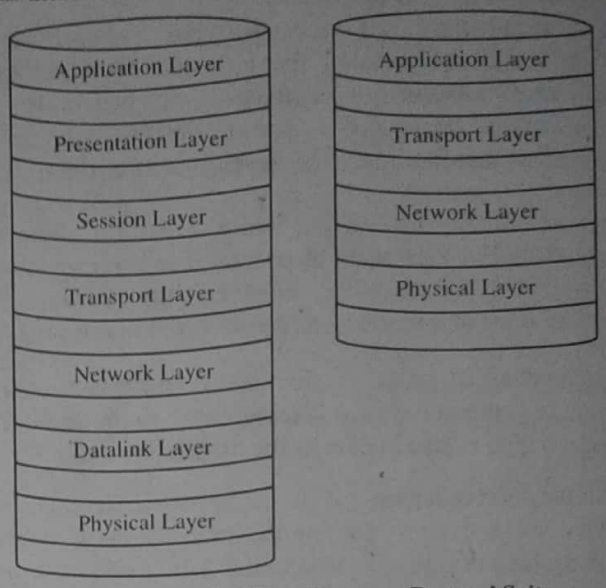


Figure 3.13: OSI and Internet Protocol Suite

4) **Data-Centred Architecture:** There are two different constituents in a data-centred architecture:

i) A central data structure or data store,(central repository), and

ii) A collection of client software.

The current state of the data is represented by the data store such as a file or database. Various operations such as add, update, delete, etc., can be performed on the data contained in the data store by the client software. There are some situations in which the client software is allowed by the data store to have access to the data though there might have been changes or actions performed by other client software.

This architectural style allows to add new components associated with the clients and modifications can be brought about in the existing components without regarding other clients as the client components operate independently.

In the blackboard system the store data has been transformed into a blackboard, notifies the client software about changes in the data because of its interest. The blackboard component also transfers the information among the clients.

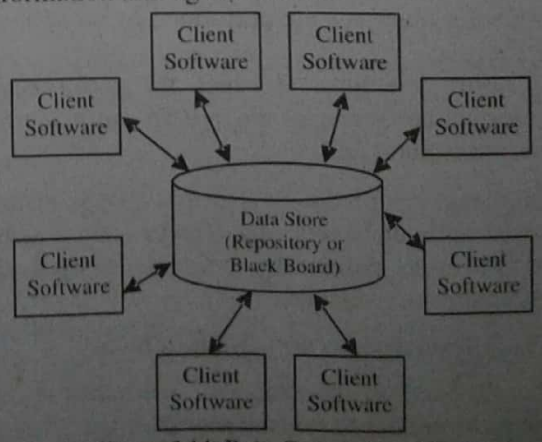


Figure 3.14: Data-Centered Architecture

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

5) **Call and Return Architecture:** An easily modifiable program structure can be obtained by software designers with the help of a call and return architecture. There are two sub-styles in this style:
   i) **Main Program/Subprogram Architecture:** A control hierarchy is created by decomposing the function. A number of program components are invoked by the main function. Other components may be further invoked by these components.

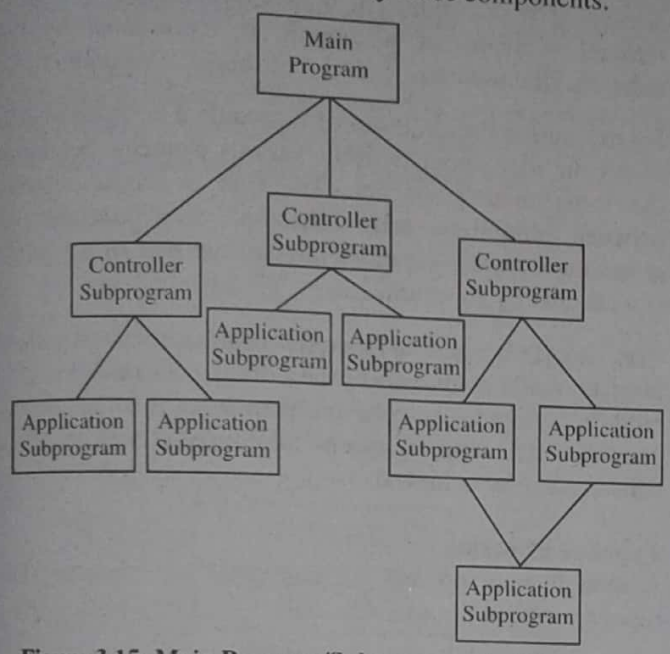


Figure 3.15: Main Program/Subprogram Architecture

   ii) **Remote Procedure Call Architecture:** Distribution of the main or subprogram architecture components is done over a network spanning over many computers.

Ques 25) What is procedural design?

Or

Write short note on effective modular design.

Ans: Procedural Design/Effective Modular Design

The aim of the procedural design is to obtain procedural description from structural elements. After the completion of architectural design and data design, procedural design starts. The algorithmic details are defined in a natural language like English.

Following are the representation of the procedures:

1) **Structured Programming:** The three main constructs that are used in structured programming are Condition, Iteration and Sequence.
2) **Graphical Design Tools:** To represent the procedure, graphical design tools like flow chart is used.
3) **Program Design Language or Structured English or Pseudo Code:** This is used for structural representation of the procedure. Subprogram, input/output condition and representation are incorporated in its syntax.

Component-level design is another name given to procedural design. To create the component-level design the structural elements defined by the software architecture are transformed into procedural descriptions of software components. The data flow-oriented element of the analysis model serves as the base of these components.

One of the following three roles is served by the component or module that is a part of the software architecture:

1) **Control Component:** It is responsible for coordinating the invocation of the components that are part of the problem domain.
2) **Problem Domain Component:** It is responsible for implementing a partial or complete function as per user requirement.
3) **Infrastructure Component:** It provides support to functions which in turn provide support to problem domain processing requirements.

The communication mechanisms, algorithms, interface description and data structures allocated to each module is described by the component-level design. A modular building block of the software is a component or module. Depending on the use by a software engineer the significance of component varies.

The following properties must be exhibited by the modular design:

1) **Provide Simple Interface:** The number of interactions needed to verify whether the system is performing the functions as desired is reduced by simple interfaces. The components can be also reused easily in different circumstances if the interfaces are simple. One of the major cost savers is reuse. The development cost can be distributed over multiple projects due to the fact that there is decrease in time spent on design, coding and testing. Various studies have shown that one of the effective ways of cutting software development cost is the reuse of software design.
2) **Ensure Information Hiding:** Merely subdividing a program does not offer the advantages of modularity. The information that is unavailable to the rest of the program must be encapsulated within a module. The cost associated with subsequent design changes is decreased due to this.

   For example, functions that are used in multiple parts of a system or which have a common implementation can be encapsulated in a module.

Many engineering disciplines have accepted the approach of modularity. There has been a notable reduction in the complexity of software design with the introduction of modular design. Parallel development of systems has been encouraged due to ease of making changes in programs.

Functional independence, a design concept, is considered to be very crucial in attaining efficient modularity.

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

Ques 26) What is functional independence? What is the need of functional independence?

Or

What do you mean functional independence? Why functional independence is the key factor for a good software design? Explain.

Or

List out the qualitative criteria for measuring function independence

Ans: Functional Independence

The notion of information hiding and abstraction has directly given advent to the concept of functional independence. In order to achieve functional independence, modules with an "aversion" to excessive interaction with other modules and 'single-minded' functions are developed. The aim with which the system is designed is that the interface is simple when a module is seen from other parts of the system and it addresses a specific sub-function of requirements. A module has very low rate of interaction with other modules if it is functionally independent.

It is easier to develop software if it has independent modules due to the fact that it may have simple interfaces and the compartmentalised functions.

If the modules are independent, the effects of code or design modification is minimum. Thus it is easy to maintain them and less error is propagated. Reuse of the modules also becomes easy. The key to good design is functional independence which is in turn the key to the quality of the software.

There are two qualitative criteria using which independence can be measured – **coupling** and **cohesion**.

Need of Functional Independence/Functional Independence as a Key Factor for a Good Software Design

One of the crucial ways to good design is functional independence because of the reasons given below:

1) **Error Isolation:** Error propagation is greatly decreased by functional independence because of the fact that it has very less communication with other modules of the software. There will be no direct effect on the other modules if there is some error in one module.
2) **Scope for Reuse:** As the interface of the module with other modules of the system is insignificant and simple and the module does some specific and well defined functions, it is possible to reuse the module.
3) **Understandability:** As the modules are independent of each other there is notable reduction in the complexity of the design. It is also possible to understand the system in isolation.

Qualitative Criteria for Measuring Functional Independence

In order to measure the functional independence there are two qualitative criteria as given below:

1) Cohesion
2) Coupling

Ques 27) What is cohesion? Also discuss its types.

Ans: Cohesion

The measure of the functional strength of a module is known as cohesion. Using cohesion an attempt is made to increase the strength of the bond amid the elements belonging to the same module by maximising the relationship between them. The intra-module relationship is ensured by cohesion. The degree of binding between the internal elements of a module is represented by the cohesion of a module.

By examining the cohesion of a module a designer is able to get the idea about whether various elements belong to the same module or not. There is a certain relation between coupling and cohesion. The strength of association among the elements that are part of a module is measured by cohesion.

Cohesion is a module specific characteristic. Attempts must be made to maximise the cohesion in modules. If a module is maximally cohesive then it tends to be loosely coupled. Thus coupling can be minimised if high levels of cohesion can be achieved.

Types of Cohesion

A module may possess various types of cohesion. The types have been discussed below:

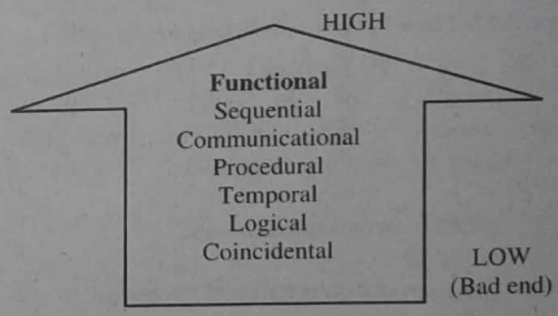


Figure 3.16: Types of Cohesion

1) **Coincidental Cohesion:** If the tasks performed by a module are very loosely related to each other than coincidental cohesion is said to exist.

Function A	
Function B	Function C
Function D	Function E

Figure 3.17: Coincidental Cohesion Parts Unrelated

A random collection of functions are found in the module in such a situation. It indicates that the functions have been included in the module without proper design or planning. This cohesion is worst in nature.

2) **Logical Cohesion:** If all the elements of the module perform like functions then it is said to be logically cohesive. **For example,** data output, data input, error handling, etc.

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

If a set of print functions responsible for the generation of output reports are arranged within the same module, they showcase the example of logical cohesion.

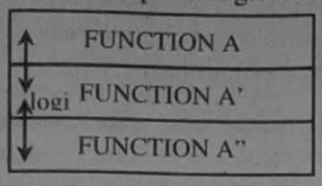


Figure 3.18: Logical Cohesion Similar Functions

3) **Temporal Cohesion:** A module is said to have temporal cohesion if the functions of the module are associated to each other due to the reason that they have to be executed together during a certain time period.

Temporal cohesion is exhibited by set of such functions that have to carry out tasks such as initialisation, start-up and shutdown of some processes.

A component can also be temporarily cohesive in case it performs multiple functions in sequence and there is a relation between the functions because of the involved timing. **For example,** the modification of the design of a certain function X is time dependent.

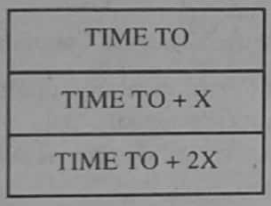


Figure 3.19: Temporal CohesionRelated by Time

4) **Procedural Cohesion:** If all the functions which are part of a set also belong to a single procedure or algorithm that has to perform certain steps in sequence in order to achieve some objective, then the module is said to possess procedural cohesion. The algorithm for message decoding is an **example** of such cohesion.

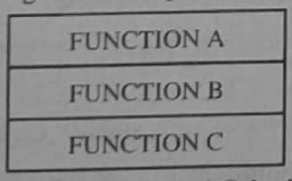


Figure 3.20: Procedural Cohesion Related by order of functions

5) **Communicational Cohesion:** If the all the functions within a module perform the task of updating the same data structure, then the module is said to have communicational cohesion. A set of functions that have been defined as a stack or array, exhibits such cohesion.

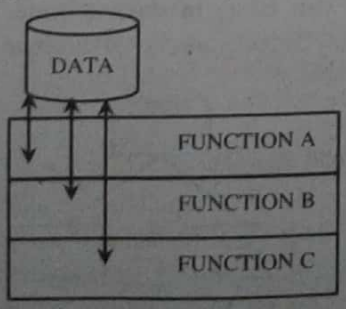


Figure 3.21: Communicational Cohesion Access Same Data

6) **Sequential Cohesion:** If the elements belonging to a module are part of a sequence in which the output produced by one element is the input to the next in the sequence then the module is said to exhibit sequential cohesion.

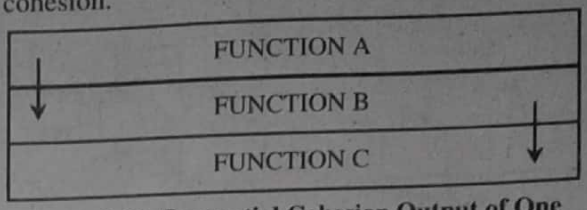


Figure 3.22: Sequential Cohesion Output of One Part is Input to Next

7) **Functional Cohesion:** In case the various elements contained in a module are working together to accomplish a common function then functional cohesion is said to happen. **It is the best cohesion.**

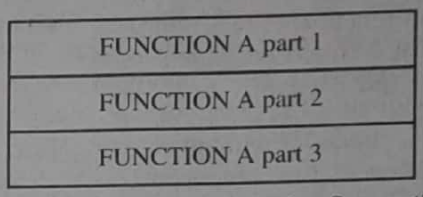


Figure 3.23: Functional Cohesion Sequential with Complete, Related Functions

For example, functional cohesion is seen in a module that comprises the functions essential to accomplish the payroll of employees. It is possible to define the role played by a module in single sentence if it exhibits functional cohesion.

Ques 28) What is coupling? Classify them.

Or

Discuss the coupling in the context of software design.

Or

What are the reasons for using low coupling for a good design?

Ans: Coupling

The extent of interdependence or the strength of interconnection amid the modules is known as coupling. If one module can perform the task completely without the existence of the other then the two are considered to be independent. It is possible to modify and resolve the two modules independently if they are independent.

It is not possible to make all the modules within a system independent of each other as they have to communicate with each other to produce the required output that is expected to the environment external to the system.

More knowledge about a module is needed in order to solve or understand it in case the dependency amid the modules increases due to more connections between them. It is possible to understand one module without understanding the other if connections between the modules are easy and less in numbers. "How strongly" various modules are connected with each other is the

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

prime notion described by coupling. This has been illustrated in **figure 3.24**.

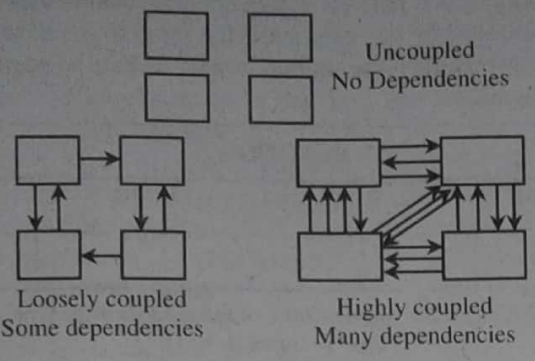


Figure 3.24: Levels of Coupling

As coupling is abstract in nature, it is difficult to quantify it. Thus there is no formula in existence to determine coupling between the modules.

The objective is to have **low coupling and high cohesion.** A good example of a system that has high cohesion and low coupling is the 'plug and play' feature of the computer system. Various slots in the motherboard of the system simply facilitate to add or remove the various services/functionalities without affecting the entire system. This is because add on components provide the services in highly cohesive manner.

Reasons for Low Coupling for a Good Design

A good design will have low coupling. Thus, interfaces should be carefully specified in order to keep low value of coupling. **For example,** coupling increases as the number of calls between increases, or the amount of shared data increases. The hypothesis is that design with high coupling will have more errors. Loose coupling, on the other hand, minimizes the interdependence amongst modules. This can be achieved in the following ways:

1) Controlling the number of parameters passed amongst modules.
2) Avoid passing undesired data at calling module
3) Maintain parent/child relationship between calling and called modules.
4) Pass data, not the control information.

Types of Coupling

The relative interdependence among modules is known as coupling. The factors on which coupling is dependents are the data passed across the interface, the point of reference or entry to the module and the complexity of interface of the modules. Priority is always given to low coupling. Following are the different types of coupling:

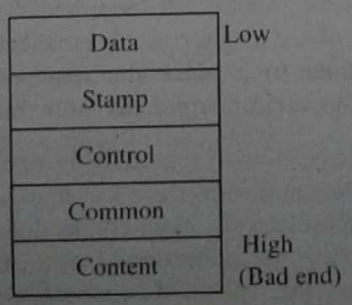


Figure 3.25: Classification of Coupling

1) **Data Coupling:** If two modules use an elementary data item to communicate among themselves then the two modules are said to be data couples. **Few example,** character, integer or float. The data item must not be used for control purpose but should be related to the problem only.

   Modules are said to be data coupled if non-global variables are passed to it. This is considered to be the lowest form of coupling. The data which has to be passed to the subordinate module must be known by the higher-level module or calling module, while the called or subordinate module should be only aware of the data which it needs and the data it has to return.

   For example, one module in C passes an 'int' variable and is received by another module by its value.

   It is the best coupling.

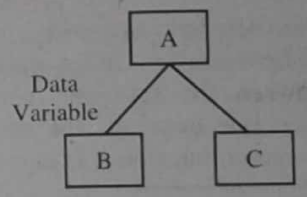


Figure 3.26: Data Coupling

2) **Stamp Coupling:** In case two modules interact with each other using composite data items like object, record, structure etc., then the modules are said to be stamp coupled.

   Modules are also said to be stamp coupled if one module passes an entire structure or non-global data structure to another module. **For example,** passing a structure variable in C, record in Pascal or object in C++.

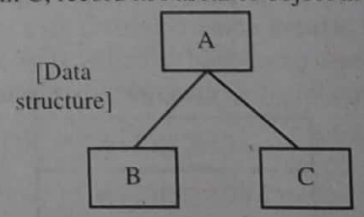


Figure 3.27: Stamp Coupling

3) **Control Coupling:** In case one module sends data to another module to direct the order in which the instructions are executed, it is known as control coupling. If a flag set in one module is tested in another module then it is an example of control coupling.

   The detailed working of the inner working of the receiving module must be known to sending module. A variable that is set in super ordinate module A is used to control decisions in subordinate module C by passing it to C.

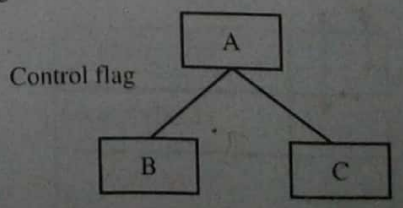


Figure 3.28: Control Coupling

DOWNLOADED FROM KTUASSIST.IN
Scanned by CamScanner

4) **Common Coupling:** If two modules share global data items then they are said to be common coupled. If there is large common coupling in modules then it becomes time consuming and difficult to diagnose the problems. This is due to the fact that if some change has to be made in the common data the software engineer has to track back to all those components that have access to that data so that evaluation of the effect of changes can be carried out.

   Determining the component that is responsible for setting a particular value for a variable becomes difficult.

5) **Content Coupling:** If the code of two modules is mutual then there is content coupling between them. **For example,** branch from one module to another module. This is considered to be the highest form of coupling and there is high interdependence among the modules. In this the inner working of one module is directly referred to by another module.

   For example, using the GOTO label D1 statement, the control is transferred from component B to D.

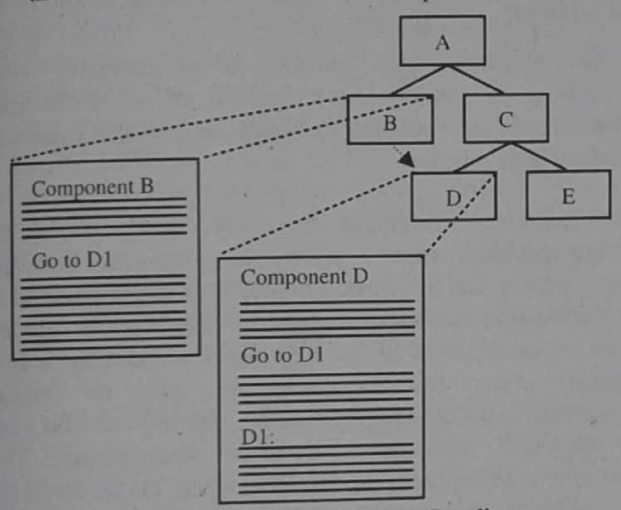


Figure 3.29: Example of Content Coupling

Ques 29) What is difference between cohesion and coupling?

Or

Differentiate between cohesion and coupling.

Ans: Difference between Cohesion and Coupling

Table below shows the comparison between cohesion and coupling:

Table 3.7: Comparison between Cohesion and Coupling

Cohesion	Coupling
Cohesion is a measure of functional strength of a module. A cohesive module performs a single task within a software procedure, requiring little interaction with procedures being performed in other parts of a program.	Coupling between modules is the strength of interconnections between modules or a measure of interdependence among modules.
Cohesion is to strengthen the bond between elements of the same module by maximizing the relationship between elements of the same module.	The notion of coupling attempts to capture this concept of "how strongly" different modules are in interconnected.
Cohesion of a module represent how tightly bound the internal elements of the module are to one another.	Coupling is a measure of the relative interdependence among modules.
The scale of cohesion is nonlinear, i.e., low-end cohesiveness is much worse than middle range (but focus on high end cohesion).	A low coupling is always preferred.
Different classes of cohesion that a module may possess are listed as: Coincidental Cohesion, Logical Cohesion, Temporal Cohesion, Procedural Cohesion, Sequential Cohesion and Functional Cohesion.	Different types of module coupling can be listed as: Data coupling, Stamp Coupling, Control Coupling, Common Coupling and Content Coupling.
For example, in highly cohesive systems functionality is strongly related.	**For example,** customer billing system.

Ques 30) What are the different software design strategies? What are the objectives of Top-Down Design Strategy?

Or

Discuss the top-down and bottom-up software design strategies.

Ans: Software Design Strategies

A system consists of components, which have components of their own; indeed a system is a hierarchy of components. The highest-level component corresponds to the total system. To design such a hierarchy, there are two possible strategies:

1) **Top-Down Design Strategy:** A top-down design strategy starts by identifying the major components of the system, decomposing them into their lower-level components and iterating until the desired level of detail is achieved.

   Top-down design methods often result in some form of stepwise refinement. Starting from an abstract design, in each step the design is refined to a more concrete level, until we reach a level where no more refinement is needed and the design can be implemented directly.

   The top-down approach has been promulgated by many researchers and has been found to be extremely

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner

useful for design. Most design methodologies are based on the top-down approach. **Figure 3.30** shows top down design strategy

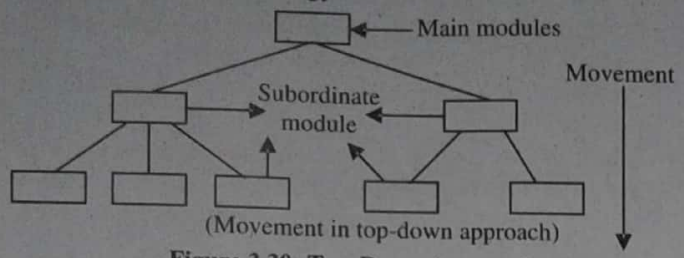


Figure 3.30: Top-Down Approach

Top-down design is a successive concretization of abstractly described ideas for a solution. Abstraction proves to be the best medium to render complex systems comprehensible; the designer is involved only with those aspects of the system that are necessary for understanding before the next step or during the search for a solution.

A top-down approach is suitable only if the specifications of the system are clearly known and the system development is from scratch. Hence, it is a reasonable approach if a waterfall type of process model is being used.

Objectives of Top-Down Design Strategy

The main objective of top-down design strategy is:

1) To systematize the design process.
2) To produce a modular program design
3) To provide a framework in which problem solving can more effectively proceed.

The main **advantage** of top-down approach is that its strong focus on specific requirements helps to make a design responsive to its requirement.

2) **Bottom-up Design Strategy:** A bottom-up design strategy starts with designing the most basic or primitive components and proceeds to higher-level components that use these lower-level components.

Bottom-up methods work with layers of abstraction. Starting from the very bottom, operations that provide a layer of abstraction are implemented. The operations of this layer are then used to implement more powerful operations and still higher layer of abstraction, until the stage is reached where the operations supported by the layer are those desired by the system.

In Bottom up design, the approach is to start at the bottom, with problems that one already knows how to solve or we have reusable software component. Then from there, we can work upwards towards a solution to the overall problem.

If a system is to be built from an existing system, a bottom-up approach is more suitable, as it starts from some existing components. So, if an iterative enhancement type of process is being followed, in later iterations, the bottom-up approach could be more suitable (in the first iteration a top-down approach can be used).

This approach is frequently used for developing systems (**figure 3.31**).

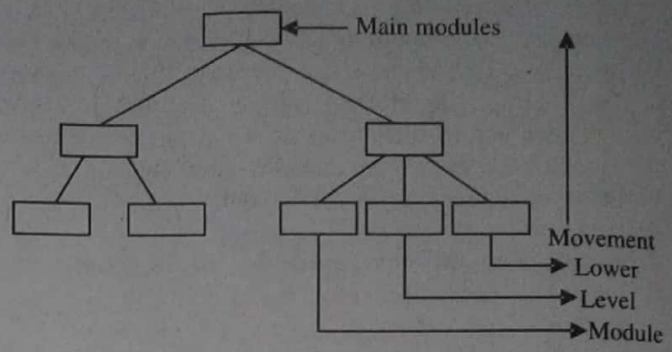


Figure 3.31: Bottom-Up Approach

Ques 31) Explain stepwise refinement in detail.

Ans: Stepwise Refinement

A top-down design strategy is referred to as stepwise refinement. By continuously filtering stages of procedural detail a program can be established. And a hierarchy is developed in a procedural abstraction where refining takes place at every level till programming language statements are achieved.

It is an elaboration process which initiates with describing of function at a high level of abstraction. More details are provided as further refinement occurs at successive levels.

The stepwise refinement technique breaks the logic design problem into a series of steps, so that the development can be done gradually. The process starts by converting the specifications of the module into an abstract description of an algorithm containing a few abstract statements. In each step, one or several statements in the algorithm developed so far are decomposed into more detailed instructions. The successive refinement terminates when all instructions are sufficiently precise that they can be easily be converted into programming language statements. During refinement, both data and instructions have to be refined. A guideline for refinement is that in each step the amount of decomposition should be such that it can be easily handled and that represents one or two design decisions.

Stepwise refinement involves the following activities:

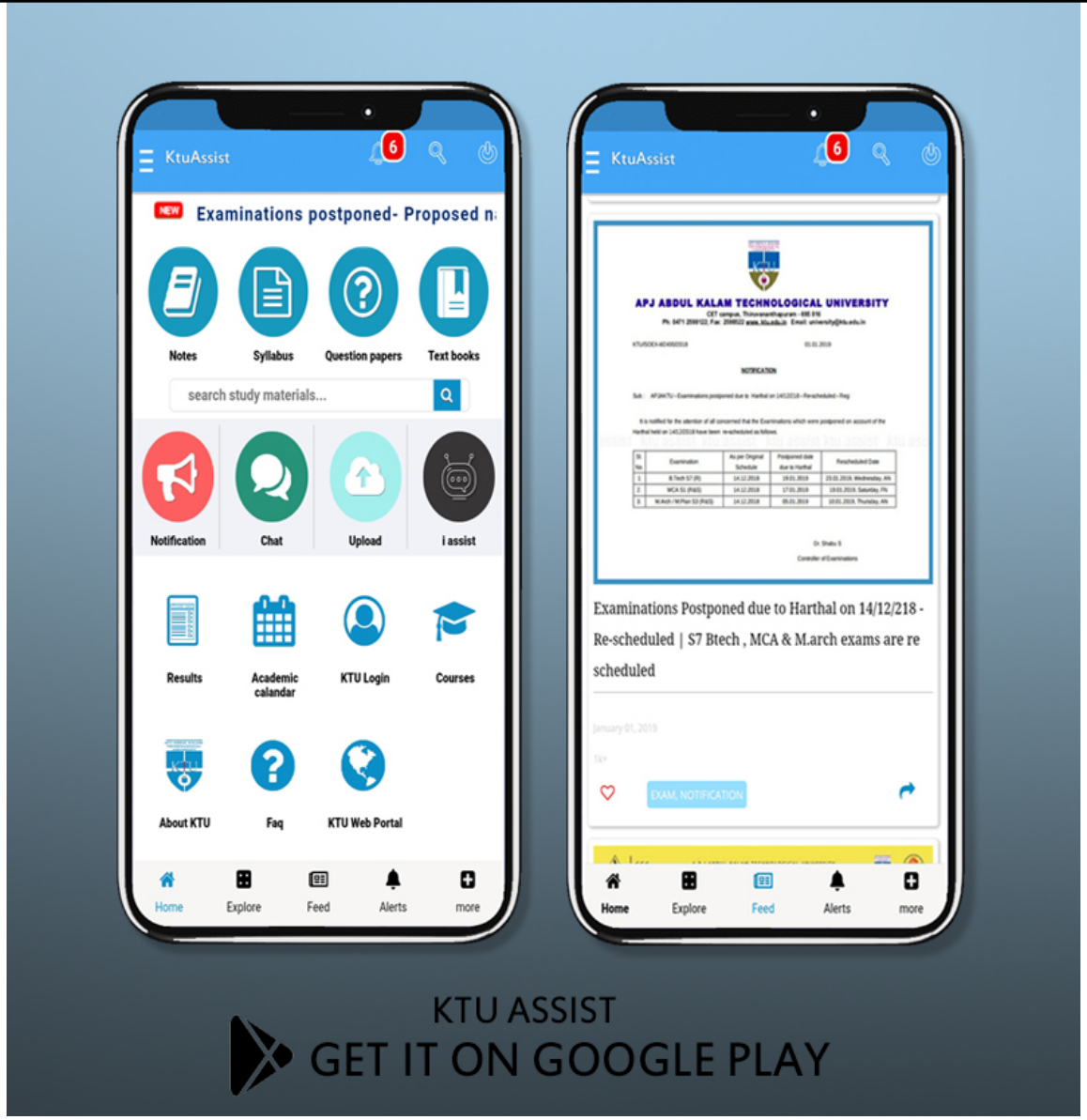
1) Isolating design aspects that are not truly interdependent.
2) Postponing decisions concerning representation details as long as possible.
3) Carefully demonstrating that each successive step in the refinement process is a faithful expansion of previous steps.

The major **benefits of stepwise refinement** as a design technique are:

1) Top-down decomposition.
2) Incremental addition of detail
3) Postponement of design decisions

DOWNLOADED FROM KTUASSIST.IN

Scanned by CamScanner



END

facebook.com/ktuassist

instagram.com/ktu_assist