Unit – III

Planning a Software Project

Cost estimation

A software cost estimating methodology is an indirect metric used by software professionals to estimate project costs. They're utilized for a variety of things. It contains the following items −

- Budgeting − the most desired capability is for the overall estimate to be correct. As a result, the first focus is on estimating the software product's budget.

- Controlling and planning the project − another option is to break down costs and schedules by component, stage, and activity.

Models of Cost Estimation

Software engineering cost models has had its share of difficulties. The fast-paced nature of software development has made developing parametric models that deliver high accuracy for software development in all disciplines very challenging. S/w development expenses are rising at an extraordinary pace, and practitioners are constantly lamenting their inability to effectively forecast the costs involved. Because most models are private, they cannot be compared and contrasted in terms of model structure. These are the following −

For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take. Several estimation procedures have been developed and are having the following attributes in common.

1. Project scope must be established in advanced.

2. The project is broken into small PCs which are estimated individually.

   To achieve true cost & schedule estimate, several option arise.

3. Delay estimation

4. Used symbol decomposition techniques to generate project cost and schedule estimates.

5. Acquire one or more automated estimation tools.

Uses of Cost Estimation

1. During the planning stage, one needs to choose how many engineers are required for the project and to develop a schedule.

2. In monitoring the project's progress, one needs to access whether the project is progressing according to the procedure and takes corrective action, if necessary.

Static, Single Variable Models: When a model makes use of single variables to calculate desired values such as cost, time, efforts, etc. is said to be a single variable model. The most common equation is:

$$C=aL^b$$

Where

C=Costs

L=size a and b are constants

The Software Engineering Laboratory established a model called SEL model, for estimating its software production. This model is an example of the static, single variable model.

$$E=1.4L^{0.93}$$

$$DOC=30.4L^{0.90}$$

$$D=4.6L^{0.26}$$

Where

E = Efforts (Person per Month)

DOC = Documentation (Number of Pages)

D = Duration (D, in months)

L = Number of Lines per code

COCOMO

(a) COCOMO fundamentals

The term COCOMO stands for Constructive Cost Model. Due to the model's ease of transparency, it provides the magnitude of the project's expense. It is designed for small projects since it has a limited number of cost drivers. When the team size is small, i.e. when the staff is tiny, it is helpful.

It's useful for getting a quick, early, rough estimate of software costs, but its accuracy is limited due to the lack of factors to account for differences in hardware constraints, personnel quality and experience, use of modern tools and techniques, and other project attributes that are known to have a significant impact on s/w costs.

EFFORT = a* (KDSI) b EFFORT = a* (KDSI)

Constants **a** and **b** have different values depending on the project type. The KLOC is the projected number of delivered lines of code for the project.

The following are the three kinds of modes available in COCOMO −

- Organic Mode − A modest, basic software project involving a small group of people with prior application knowledge. Efforts, E, and Development, D are the following −

$$E = 2.4*(KLOC)^{1.05}$$

$$D=2.5*(E)^{0.38}$$

- Semi-detached Mode − an intermediate software project in which teams with varying levels of expertise collaborate.

$$E= 3.0*(KLOC)^{1.12}$$

$$D=2.5*(E)^{0.35}$$

- Embedded Mode − A software project that must be built under strict hardware, software, and operational limitations.

$$E= 3.6*(KLOC)^{1.20}$$

$$D= 2.5*(E)^{0.32}$$

(b) COCOMO Intermediate

It assesses software development effort as a function of program size and a set of cost drivers, which include a subjective assessment of goods, hardware, employees, and project characteristics.

It's appropriate for medium-sized tasks. Intermediate to basic and advanced COCOMO are the cost drivers. Cost factors influence product dependability, database size, execution, and storage. The team has a modest size. The COCOMO intermediate model looks like this −

EFFORT = a*(KLOC) b*EAF

Here, effort is measured in person-months, and KLOC is the project's projected amount of delivered lines of code.

MODEL PUTNAM (SLIM)

The Software Life Cycle Model (SLIM) is based on Putnam's research on the Rayleigh distribution of project staff level vs time. It combines the majority of common size estimation approaches, such as ballpark techniques, source instructions, function pointers, and so on. It calculates the project's work, timeline, and defect rate. Data from previously completed projects is collected and analyzed, and the model is then standardized. If data is not available, a series of questions may be completed to acquire MBI and PF values from the database. P stands for productivity, which is defined as the ratio of software product size S to development effort E, as follows −

$$S/E = P$$

Techniques for Cost Estimation

A. Algorithmic Methodologies

The software cost estimate is calculated using an algorithmic technique, which uses a formula. The formula is derived from cost-factor models that are formed by merging them. Additionally, the statistical approach is employed to build the model. The algorithmic technique is intended to give a set of mathematical equations that may be used to estimate software. These mathematical calculations are based on research and historical data, and they take into account factors like the amount of source lines of code (SLOC), the number of functions to run, and other cost drivers like language, design approach, talent levels, risk assessments, and so on. Many models, such as COCOMO models, Putnam models, and function points based models, have been built using algorithmic approaches that have been extensively investigated.

Analysis of Function Points

Another approach of assessing the size and complexity of a software system in terms of the services it provides to the user is the Function Point Analysis.

Counting function points involves two steps

- Compiling a list of user functions − The raw function counts are calculated using a linear combination of five fundamental software components: external inputs, external outputs, external queries, logic internal files, and external interfaces, all of which are categorized into three degrees of complexity: simple, average, and difficult. The number of function counts is the total of these numbers, weighted according to the difficulty level (FC).

- Top-Down Method of Estimation

Macro Model is another name for top-down estimation. The overall cost estimate for the project is obtained from the global attributes of the software project using the top-down estimating approach, and then the project is partitioned into several low-level mechanisms or components. The Putnam model is the most popular technique that employs this strategy. When only global attributes are known, this technique is better appropriate for early cost estimate. Because there is no precise information accessible in the early stages of software development, it is quite valuable.

- Bottom-Up Method of Estimation

The cost of each software component is calculated using the bottom-up estimating approach, and the findings are then combined to arrive at an estimated cost of the total project. Its goal is to build a system estimate using the information gathered about minor software components and their interactions. COCOMO's detailed model is the most popular technique that employs this methodology.

Software Project size estimation techniques

Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation. Some of these are:

- Lines of Code
- Number of entities in ER diagram
- Total number of processes in detailed data flow diagram
- Function points

1. Lines of Code (LOC): As the name suggest, LOC count the total number of lines of source code in a project. The units of LOC are:

- KLOC- Thousand lines of code
- NLOC- Non comment lines of code
- KDSI- Thousands of delivered source instruction

The size is estimated by comparing it with the existing systems of same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

Advantages:

- Universally accepted and is used in many models like COCOMO.
- Estimation is closer to developer’s perspective.

- Simple to use.

Disadvantages:

- Different programming languages contains different number of lines.
- No proper industry standard exist for this technique.
- It is difficult to estimate the size using this technique in early stages of project.

2. Number of entities in ER diagram: ER model provides a static view of the project. It describes the entities and its relationships. The number of entities in ER model can be used to measure the estimation of size of project. Number of entities depends on the size of the project. This is because more entities needed more classes/structures thus leading to more coding.

Advantages

- Size estimation can be done during initial stages of planning.
- Number of entities is independent of programming technologies used.

Disadvantages:

- No fixed standards exist. Some entities contribute more project size than others.
- Just like FPA, it is less used in cost estimation model. Hence, it must be converted to LOC.

3. Total number of processes in detailed data flow diagram: Data Flow Diagram(DFD) represents the functional view of a software. The model depicts the main processes/functions involved in software and flow of data between them. Sum of the estimated size of each process gives the final estimated size.

Advantages:

- It is independent of programming language.
- Each major processes can be decomposed into smaller processes. This will increase the accuracy of estimation

Disadvantages:

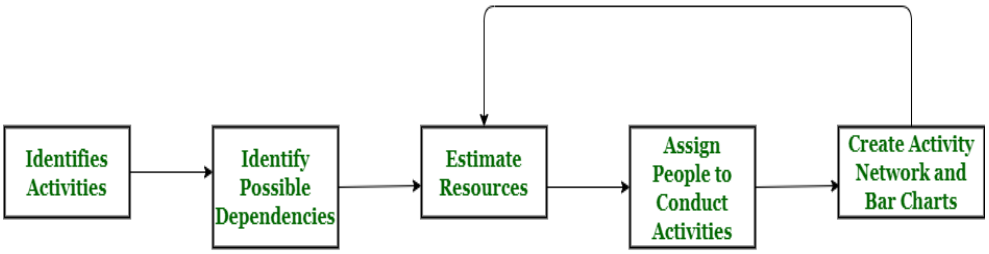
- Studying similar kind of processes to estimate size takes additional time and effort.
- All software projects are not required to construction of DFD.

4. Function Point Analysis: In this method, the number and type of functions supported by the software are utilized to find FPC(function point count).

Project Scheduling

A schedule in your project's time table actually consists of sequenced activities and milestones that are needed to be delivered under a given period of time.

Project schedule simply means a mechanism that is used to communicate and know about that tasks are needed and has to be done or performed and which organizational resources will be given or allocated to these tasks and in what time duration or time frame work is needed to be performed. Effective project scheduling leads to success of project, reduced cost, and increased customer satisfaction. Scheduling in project management means to list out activities, deliverables, and milestones within a project that are delivered. The most common and important form of project schedule is Gantt chart.



Project Scheduling Process

Process:

The manager needs to estimate time and resources of project while scheduling project. All activities in project must be arranged in a coherent sequence that means activities should be arranged in a logical and well-organized manner for easy to understand.

The total work is separated or divided into various small activities or tasks during project schedule. Then, Project manager will decide time required for each activity or task to get completed. Even some activities are conducted and performed in parallel for efficient performance. The project manager should be aware of fact that each stage of project is not problem-free.

Resources required for Development of Project:

- Human effort
- Sufficient disk space on server
- Specialized hardware
- Software technology
- Travel allowance required by project staff, etc.

Advantages of Project Scheduling:

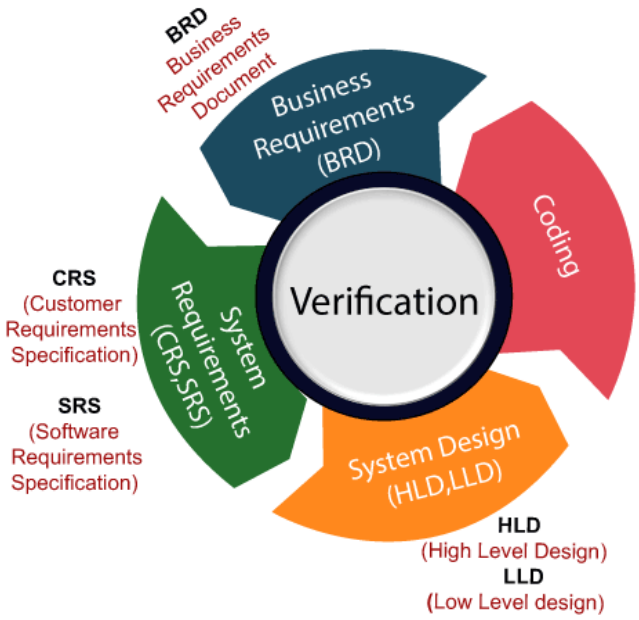
There are several advantages provided by project schedule in our project management:

- It simply ensures that everyone remains on same page as far as tasks get completed, dependencies, and deadlines.
- It helps in identifying issues early and concerns such as lack or unavailability of resources.
- It also helps to identify relationships and to monitor process.
- It provides effective budget management and risk mitigation.

Project-task scheduling is a significant project planning activity. It comprises deciding which functions would be taken up when. To schedule the project plan, a software project manager wants to do the following:

1. Identify all the functions required to complete the project.
2. Break down large functions into small activities.
3. Determine the dependency among various activities.
4. Establish the most likely size for the time duration required to complete the activities.
5. Allocate resources to activities.
6. Plan the beginning and ending dates for different activities.
7. Determine the critical path. A critical way is the group of activities that decide the duration of the project.

Verification and Validation Testing



Verification testing

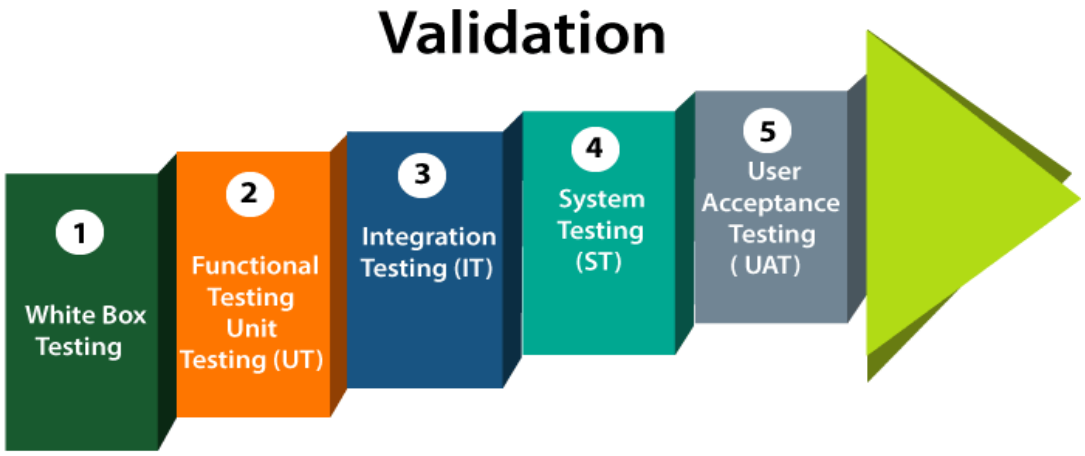
Verification testing includes different activities such as business requirements, system requirements, design review, and code walkthrough while developing a product.

It is also known as static testing, where we are ensuring that "**we are developing the right product or not**". And it also checks that the developed application fulfilling all the requirements given by the client.

Validation testing

Validation testing is testing where tester performed functional and non-functional testing. Here **functional testing** includes Unit Testing (UT), Integration Testing (IT) and System Testing (ST), and **non-functional** testing includes User acceptance testing (UAT).

Validation testing is also known as dynamic testing, where we are ensuring that **"we have developed the product right."** And it also checks that the software meets the business needs of the client.

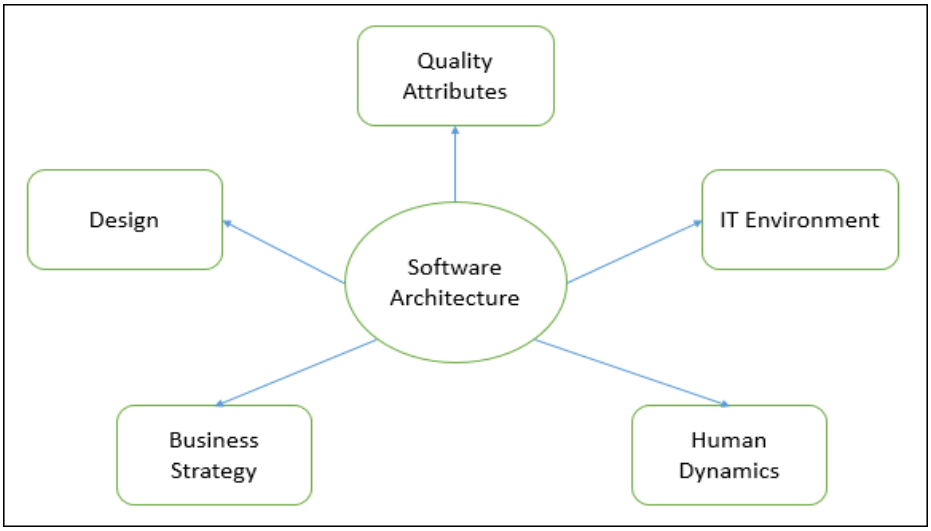


Difference between verification and validation testing

Verification	Validation
We check whether we are developing the right product or not.	We check whether the developed product is right.
Verification is also known as **static testing**.	Validation is also known as **dynamic testing**.
Verification includes different methods like Inspections, Reviews, and Walkthroughs.	Validation includes testing like functional testing , system testing, integration , and User acceptance testing.
Quality assurance comes under verification testing.	**Quality control** comes under validation testing.
The execution of code does not happen in the verification testing.	In validation testing, the execution of code happens.
Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements.	Validation testing is executed by the testing team to test the application.
Verification is done before the validation testing.	After verification testing, validation testing takes place.

Software Architecture

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, non-functional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

Software Architecture

Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

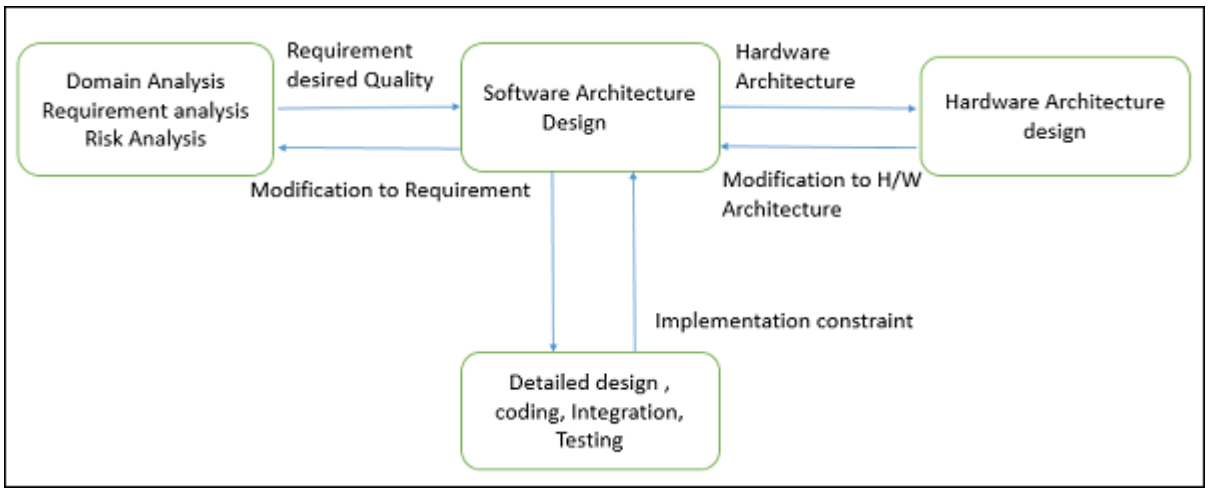
- It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
- Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of −
    - Selection of structural elements and their interfaces by which the system is composed.
    - Behavior as specified in collaborations among those elements.
    - Composition of these structural and behavioral elements into large subsystem.
    - Architectural decisions align with business objectives.
    - Architectural styles guide the organization.

Software Design

Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfil the requirement of the system. The objectives of having a design plan are as follows −

- To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.
- Act as a blueprint during the development process.
- Guide the implementation tasks, including detailed design, coding, integration, and testing.

It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.



Role of Software Architect

A Software Architect provides a solution that the technical team can create and design for the entire application. A software architect should have expertise in the following areas −

Design Expertise

- Expert in software design, including diverse methods and approaches such as object-oriented design, event-driven design, etc.
- Lead the development team and coordinate the development efforts for the integrity of the design.
- Should be able to review design proposals and trade-off among themselves.

Domain Expertise

- Expert on the system being developed and plan for software evolution.
- Assist in the requirement investigation process, assuring completeness and consistency.
- Coordinate the definition of domain model for the system being developed.

Technology Expertise

- Expert on available technologies that helps in the implementation of the system.
- Coordinate the selection of programming language, framework, platforms, databases, etc.

Methodological Expertise

- Expert on software development methodologies that may be adopted during SDLC (Software Development Life Cycle).
- Choose the appropriate approaches for development that helps the entire team.

Hidden Role of Software Architect

- Facilitates the technical work among team members and reinforcing the trust relationship in the team.
- Information specialist who shares knowledge and has vast experience.
- Protect the team members from external forces that would distract them and bring less value to the project.

Introduction to OO Paradigm

OO paradigm is a significant methodology for the development of any software. Most of the architecture styles or patterns such as pipe and filter, data repository, and component-based can be implemented by using this paradigm.

Basic concepts and terminologies of object–oriented systems −

Object

An object is a real-world element in an object–oriented environment that may have a physical or a conceptual existence. Each object has −

- Identity that distinguishes it from other objects in the system.
- State that determines characteristic properties of an object as well as values of properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modelled according to the needs of the application. An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

Class

A class represents a collection of objects having same characteristic properties that exhibit common behavior. It gives the blueprint or the description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, an object is an **instance** of a class.

The constituents of a class are −

- A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.
- A set of operations that portray the behavior of the objects of the class. Operations are also referred as functions or methods.

Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. It permits the elements of the class to be accessed from outside only through the interface provided by the class.

Polymorphism

Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instances they are operating upon. Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing inheritance.

Inheritance

It is a mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses.

The subclass can inherit or derive the attributes and methods of the super-class (es) provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines a “is – a” relationship.

OO Analysis

In object-oriented analysis phase of software development, the system requirements are determined, the classes are identified, and the relationships among classes are acknowledged. The aim of OO analysis is to understand the application domain and specific requirements of the system. The result of this phase is requirement specification and initial analysis of logical structure and feasibility of a system.

The three analysis techniques that are used in conjunction with each other for object-oriented analysis are object modelling, dynamic modelling, and functional modelling.

Object Modelling

Object modelling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

The process of object modelling can be visualized in the following steps −

- Identify objects and group into classes
- Identify the relationships among classes
- Create a user object model diagram
- Define a user object attributes
- Define the operations that should be performed on the classes

Functional Modelling

Functional Modelling is the final component of object-oriented analysis. The functional model shows the processes that are performed within an object and how the data changes, as it moves between methods. It specifies the meaning of the operations of an object modelling and the actions of a dynamic modelling. The functional model corresponds to the data flow diagram of traditional structured analysis.

The process of functional modelling can be visualized in the following steps −

- Identify all the inputs and outputs
- Construct data flow diagrams showing functional dependencies
- State the purpose of each function

- Identify the constraints
- Specify optimization criteria

Object-Oriented Design

After the analysis phase, the conceptual model is developed further into an object-oriented model using object-oriented design (OOD). In OOD, the technology-independent concepts in the analysis model are mapped onto implementing classes, constraints are identified, and interfaces are designed, resulting in a model for the solution domain. The main aim of OO design is to develop the structural architecture of a system.

The stages for object–oriented design can be identified as −

- Defining the context of the system
- Designing the system architecture
- Identification of the objects in the system
- Construction of design models
- Specification of object interfaces

OO Design can be divided into two stages − Conceptual design and Detailed design.

Conceptual design

In this stage, all the classes are identified that are needed for building the system. Further, specific responsibilities are assigned to each class. Class diagram is used to clarify the relationships among classes, and interaction diagram are used to show the flow of events. It is also known as **high-level design**.

Detailed design

In this stage, attributes and operations are assigned to each class based on their interaction diagram. State machine diagram are developed to describe the further details of design. It is also known as **low-level design**.

Design Principles

Following are the major design principles −

Principle of Decoupling

It is difficult to maintain a system with a set of highly interdependent classes, as modification in one class may result in cascading updates of other classes. In an OO design, tight coupling can be eliminated by introducing new classes or inheritance.

Ensuring Cohesion

A cohesive class performs a set of closely related functions. A lack of cohesion means — a class performs unrelated functions, although it does not affect the operation of the whole system. It makes the entire structure of software hard to manage, expand, maintain, and change.