**Unit-3 (Design Engineering)**

The design phase of software development deals with transforming the customer requirements as described in the SRS document in to a form implementable using a programming language. The software design process can be divided into the following three levels or phases of design:

**1 Interface Design**

**2Architectural Design**

**3 Detailed Design**

**I/P Software requirement Specification(SRS)**

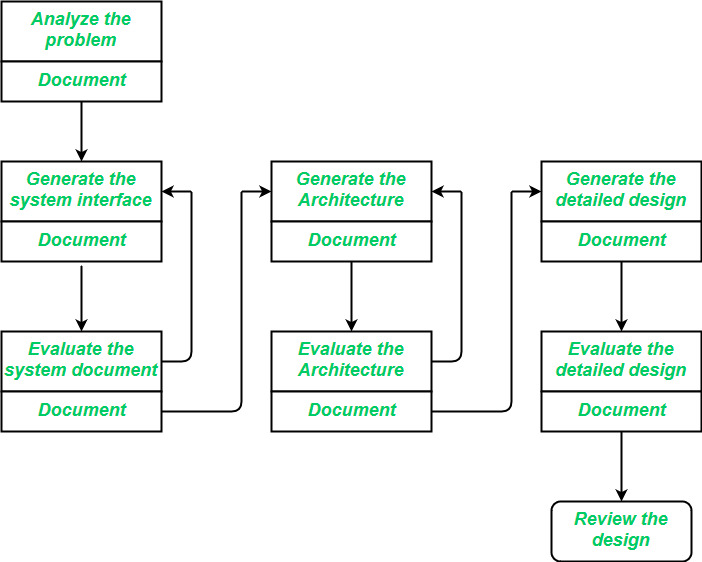
Design phase

**O/P Software design Document (SDD)**

A software design document (SDD) is a written description of a software product, that a software designer writes in order to give a software development team overall guidance to the architecture of the software project

**Elements of a System**

1. **Architecture:**This is the conceptual model that defines the structure, behavior, and views of a system. We can use flowcharts to represent and illustrate the architecture.
2. **Modules:**These are components that handle one specific task in a system. A combination of the modules makes up the system.
3. **Components:**This provides a particular function or group of related functions. They are made up of modules.
4. **Interfaces:**This is the shared boundary across which the components of a system exchange information and relate.
5. **Data:**This is the management of the information and data flow.



**Fig: Design Process**

**Interface Design**

Interface design is the specification of the interaction between a system and its environment. This phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e, during interface design, the internal of the systems are completely ignored, and the system is treated as a black box. Attention is focused on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called agents.

Interface design should include the following details:

1. Precise description of events in the environment, or messages from agents to which the system must respond.
2. Precise description of the events or messages that the system must produce.
3. Specification of the data, and the formats of the data coming into and going out of the system.
4. Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

**Architectural Design**

Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored. Issues in architectural design includes:

The architecture of software tells about the entire structure of the software. Architectural design

describes how the components of software interact with each other.

In this, different types of components communicate with each other. In this, all types of

components share or exchange information or data with each other.

In architectural design, the overall structure of the system is chosen, but the internal details of

major components are ignored.

The architecture of software tells about the entire structure of the software. Architectural design

describes how the components of software interact with each other.

In this, different types of components communicate with each other. In this, all types of

components share or exchange information or data with each other.

In architectural design, the overall structure of the system is chosen, but the internal details of

major components are ignored.

The architecture of software tells about the entire structure of the software. Architectural design

describes how the components of software interact with each other.

In this, different types of components communicate with each other. In this, all types of

components share or exchange information or data with each other.

In architectural design, the overall structure of the system is chosen, but the internal details of

major components are ignored.

1. Gross decomposition of the systems into major components.
2. Allocation of functional responsibilities to components.
3. Component Interfaces.
4. Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
5. Communication and interaction between components.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.

**Detailed Design**

Detailed design is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures. The detailed design may include:

1. Decomposition of major system components into program units.
2. Allocation of functional responsibilities to units.
3. User interfaces.
4. Unit states and state changes.
5. Data and control interaction between units.
6. Data packaging and implementation, including issues of scope and visibility of program elements.
7. Algorithms and data structures.

**Design Quality**

**1 Portability:** A software is claimed to be transportable, if it may be simply created to figure in several package environments, in several machines, with alternative code merchandise, etc.

1. **Usability:** A software has smart usability if completely different classes of users .
2. **Reusability:** A software has smart reusability if completely different modules of the merchandise will simply be reused to develop new merchandise.
3. **Correctness:** Software is correct if completely different needs as laid out in the SRS document are properly enforced.
4. **Maintainability:** A software is reparable, if errors may be simply corrected as and once they show up, new functions may be simply added to the merchandise, and therefore the functionalities of the merchandise may be simply changed, etc
5. **Reliability:**Software is more reliable if it has fewer failures. Since software engineers do not deliberately plan for their software to fail, reliability depends on the number and type of mistakes they make. Designers can improve reliability by ensuring the software is easy to implement and change, by testing it thoroughly, and also by ensuring that if failures occur, the system can handle them or can recover easily.
6. **Efficiency.** The more efficient software is, the less it uses of CPU-time, memory, disk space, network bandwidth and other resources. This is important to customers in order to reduce their costs of running the software, although with today’s powerful computers, CPU time, memory and disk usage are less of a concern than in years gone by.

**Software Design Concepts**

The **software design concept** simply means the idea or principle behind the design. It describes how you plan to solve the problem of designing software, and the logic, or thinking behind how you will design software. It allows the software engineer to create the model of the system software or product that is to be developed or built. The software design concept provides a supporting and essential structure or model for developing the right software. There are many concepts of software design and some of them are given below:

**Points to be Considered While Designing Software**

1. **Abstraction**(**Hide Irrelevant data**)**:**Abstraction simply means to hide the details to reduce complexity and increase efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution.
2. **Modularity (subdivide the system):**Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means subdividing a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. It is necessary to divide the software into components known as modules because nowadays, there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity in design has now become a trend and is also important. If the system contains fewer components then it would mean the system is complex which requires a lot of effort (cost) but if we can divide the system into components then the cost would be small.
3. **Architecture (design a structure of something):**Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.
4. **Refinement (removes impurities):**Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is a process of developing or presenting the software or system in a detailed manner which means elaborating a system or software. Refinement is very necessary to find out any error if present and then to reduce it.
5. **Pattern (a Repeated form):**A pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.
6. **Information Hiding (Hide the Information**)**:**Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and can’t be accessed by any other modules.
7. **Refactoring (Reconstruct something):**Refactoring simply means reconstructing something in such a way that it does not affect the behavior of any other features. Refactoring in software design means reconstructing the design to reduce complexity and simplify it without impacting the behavior or its functions. Fowler has defined refactoring as “the process of changing a software system in a way that it won’t impact the behavior of the design and improves the internal structure”.

**8 Coupling and Cohesion** are two key concepts in software engineering that are used to measure the quality of a software system’s design.

**Coupling** refers to the degree of interdependence between modules.

**Cohesion** refers to the degree to which elements within a module work together to fullfill a sinle and well defined task.

**Design Models**

The design phase of software Development ,transforming the customer requirements as described in the SRS documents in to design forms

Designing a model is an important phase and is a multi process that represent the data structure ,program structure ,interface characteristic and procedural Details.

There are four types of design models.

1. Data Design model
2. User Interface Design model
3. Architectural Design model
4. Component level Design model
5. **Data Design Model**
   * Represents the data objects and their interrelationships using an **entity-relationship diagram**.
   * Defines the information required for each entity or data object.
   * Illustrates the structure of data in terms of tables.
   * Depicts three types of relationships: one-to-one, one-to-many, and many-to-many.

Data design encompasses the manner in which the data objects are realized within the design. It helps to simplify the program flow, makes the design and implementation of the software components easier, and makes overall processing more efficient.

1. **User Interface (UI) Design Model**

User interface is the front-end application view to which user interacts in order to use the software. The software becomes more popular if its user interface is:

 Attractive

 Simple to use

 Responsive in short time

 Clear to understand

 Consistent on all interface screens

There are two types of User Interface:

1. **Command Line Interface:** Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.

2. **Graphical User Interface:** Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.

1. **Component-Level Design** **Model**
   * Specifies the details of individual components.
   * Addresses procedural aspects, data structures, and program structures.
   * Ensures that each component performs its intended function effectively.
2. **Architectural Design Model**
   * Focuses on the relationship between major structural elements of the software.
   * Involves decomposing the system into interacting components.
   * Expressed as a **block diagram** that provides an overview of the system structure.
   * Describes features of the components and how they communicate to share data.

# Architectural Design

**Introduction:** The software needs the architectural design to represents the design of software. IEEE defines architectural design as “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.” The software that is built for computer-based systems can exhibit one of these many architectural styles.   
Each style will describe a system category that consists of: 

* A set of components(eg: a database, computational modules) that will perform a function required by the system.
* The set of connectors will help in coordination, communication, and cooperation between the components.
* Conditions that how components can be integrated to form the system.
* Semantic models that help the designer to understand the overall properties of the system.

**Software Architecture :**  
Software Architecture defines fundamental organization of a system and more simply defines a structured solution. It defines how components of a software system are assembled, their relationship and communication between them. It serves as a blueprint for software application and development basis for developer team.

Software architecture defines a list of things which results in making many things easier in the software development process.

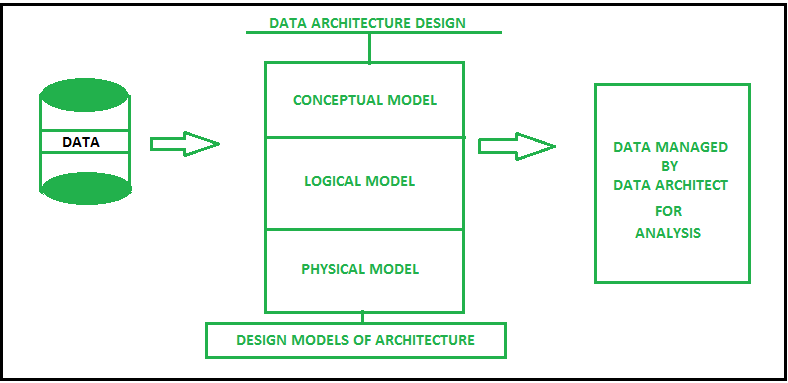
* A software architecture defines structure of a system.
* A software architecture defines behavior of a system.
* A software architecture defines component relationship.
* A software architecture defines communication structure.
* A software architecture balances stakeholder’s needs.
* A software architecture influences team structure.
* A software architecture focuses on significant elements.
* A software architecture captures early design decisions.

**Data architecture Design**   
Data architecture design is set of standards which are composed of certain policies, rules, models and standards which manages, what type of data is collected, from where it is collected, the arrangement of collected data, storing that data, utilizing and securing the data into the systems and data warehouses for further analysis.

Data is one of the essential pillars of enterprise architecture through which it succeeds in the execution of business strategy.

**It** is important for creating a vision of interactions occurring between data systems, like for example if data architect wants to implement data integration, so it will need interaction between two systems and by using data architecture the visionary model of data interaction during the process can be achieved.

Data architecture also describes the type of data structures applied to manage data and it provides an easy way for data preprocessing. The data architecture is formed by dividing into three essential models and then are combined :



* **Conceptual model –**  
  It is a business model which uses [Entity Relationship (ER) model](https://www.geeksforgeeks.org/introduction-of-er-model/) for relation between entities and their attributes.
* **Logical model –**  
  It is a model where problems are represented in the form of logic such as rows and column of data, classes, xml tags and other DBMS techniques.
* **Physical model –**  
  Physical models holds the database design like which type of database technology will be suitable for architecture.

**Architectural Style**

Each style describe a system category that encompasses

1. **A set of components**(eg. Database,Computational model) that perform a function required by a system .
2. **Set of connectors** that enable”communication ,coordination and corporation among components”.
3. **Constrain**t that define how components can be integrated to form the system.
4. **Semantic Model** that enable a designer to understand the overall functionality of a system by analyzing the known properties of its constituent’s parts.

**Taxonomy of Architectural styles:**

1. **Data centred architectures:**

 A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.

 The figure illustrates a typical data centered style. The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.

 This data-centered architecture will promote integrability. This means that the existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients.

 Data can be passed among clients using blackboard mechanism.

**Data flow architectures:** 

* This kind of architecture is used when input data is transformed into output data through a series of computational manipulative components.
* The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by lines.
* Pipes are used to transmitting data from one component to the next.
* Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don’t require any knowledge of the working of neighboring filters.
* If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.

**Call and Return architectures:** It is used to create a program that is easy to scale and modify. Many sub-styles exist within this category. Two of them are explained below.

**Object Oriented architecture:** The components of a system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing.

**Layered architecture:** 

* A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
* At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
* Intermediate layers to utility services and application software functions.
* One common example of this architectural style is OSI-ISO (Open Systems Interconnection-International Organisation for Standardisation) communication system.

***:***

**Architectural Patterns**

The architectural pattern shows how a solution can be used to solve a reoccurring problem. In another word, it reflects how a code or components interact with each other. Moreover, the architectural pattern is describing the architectural style of our system and provides solutions for the issues in our architectural style.Define architectural patterns as a way to implement our architectural style. For example:

How to separate the UI of the data module in our architectural style?

How to integrate a third-party component with our system?

How many tires will we have in our client-server architecture?

Examples of architectural patterns are microservices, message bus, service requester/ consumer, MVC, MVVM, microkernel, n-tier, domain-driven design, and presentation-abstraction-control.

Unified Modeling Language (UML)

**Unified Modeling Language (UML)** is a general purpose modelling language. The main aim of UML is to define a standard way to **visualize** the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

UML is **not a programming language**, it is rather a visual language. We use UML diagrams to portray the **behavior and structure** of a system. UML helps software engineers, businessmen and system architects with modelling, design and analysis. The Object Management Group (OMG) adopted Unified Modelling Language as a standard in 1997. Its been managed by OMG ever since. International Organization for Standardization (ISO) published UML as an approved standard in 2005. UML has been revised over the years and is reviewed periodically.

UML is linked with **object oriented** design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

1. **Structural Diagrams –** Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams , sequence diagram

2. **Behavior Diagrams –** Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

**Object Oriented Concepts Used in UML –**

1. **Class –** A class defines the blue print i.e. structure and functions of an object.

2. **Objects –** Objects help us to decompose large systems and help us to modularize our system. Modularity helps to divide our system into understandable components so that we can build our system piece by piece. An object is the fundamental unit (building block) of a system which is used to depict an entity.

3. **Inheritance –** Inheritance is a mechanism by which child classes inherit the properties of their parent classes.

4. **Abstraction –** Mechanism by which implementation details are hidden from user.

5. **Encapsulation –** Binding data together and protecting it from the outer world is referred to as encapsulation.

6. **Polymorphism –** Mechanism by which functions or entities are able to exist in different forms.

**Structural UML Diagrams –**

. **Class Diagram –** The most widely use UML diagram is the class diagram. It is the building block of all object oriented software systems. We use class diagrams to depict the static structure of a system by showing system’s classes,their methods and attributes. Class diagrams also help us identify relationship between different classes or objects.

**Component Diagram –** Component diagrams are used to represent the how the physical components in a system have been organized. We use them for modelling implementation details. Component Diagrams depict the structural relationship between software system elements and help us in understanding if functional requirements have been covered by planned development. Component Diagrams become essential to use when we design and build complex systems. Interfaces are used by components of the system to communicate with each other.

**Component Diagram –** Component diagrams are used to represent the how the physical components in a system have been organized. We use them for modelling implementation details. Component Diagrams depict the structural relationship between software system elements and help us in understanding if functional requirements have been covered by planned development. Component Diagrams become essential to use when we design and build complex systems.

**Behavioural Diagram**

**Use Case Diagrams –** Use Case Diagrams are used to depict the functionality of a system or a part of a system. They are widely used to illustrate the functional requirements of the system and its interaction with external agents(actors). A use case is basically a diagram representing different scenarios where the system can be used. A use case diagram gives us a high level view of what the system or a part of the system does without going into implementation details.

**Sequence Diagram –** A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

**Communication Diagram –** A Communication Diagram(known as Collaboration Diagram ) is used to show sequenced messages exchanged between objects. A communication diagram focuses primarily on objects and their relationships. We can represent similar information using Sequence diagrams,however, communication diagrams represent objects and links in a free form.