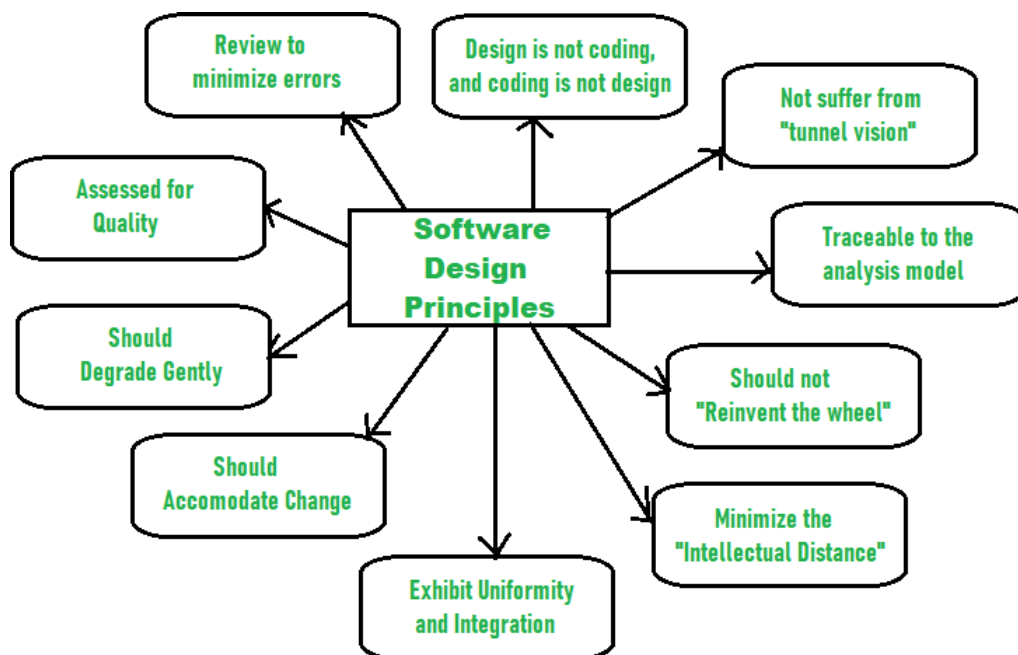


Unit III SOFTWARE DESIGN

1. Describe the Software Design Principles

Software Design is also a process to plan or convert the software requirements into a step that are needed to be carried out to develop a software system. There are several principles that are used to organize and arrange the structural components of Software design. Software Designs in which these principles are applied affect the content and the working process of the software from the beginning.



Principles Of Software Design :

1. Should not suffer from "Tunnel Vision" –

While designing the process, it should not suffer from "tunnel vision" which means that it should not only focus on completing or achieving the aim but on other effects also.

2. Traceable to analysis model –

The design process should be traceable to the analysis model which means it

should satisfy all the requirements that software requires to develop a high-quality product.

3. Should not “Reinvent The Wheel” –

The design process should not reinvent the wheel that means it should not waste time or effort in creating things that already exist. Due to this, the overall development will get increased.

4. Minimize Intellectual distance –

The design process should reduce the gap between real-world problems and software solutions for that problem meaning it should simply minimize intellectual distance.

5. Exhibit uniformity and integration –

The design should display uniformity which means it should be uniform throughout the process without any change. Integration means it should mix or combine all parts of software i.e. subsystems into one system.

6. Accommodate change –

The software should be designed in such a way that it accommodates the change implying that the software should adjust to the change that is required to be done as per the user's need.

7. Degrade gently –

The software should be designed in such a way that it degrades gracefully which means it should work properly even if an error occurs during the execution.

8. Assessed or quality –

The design should be assessed or evaluated for the quality meaning that during the evaluation, the quality of the design needs to be checked and focused on.

9. Review to discover errors –

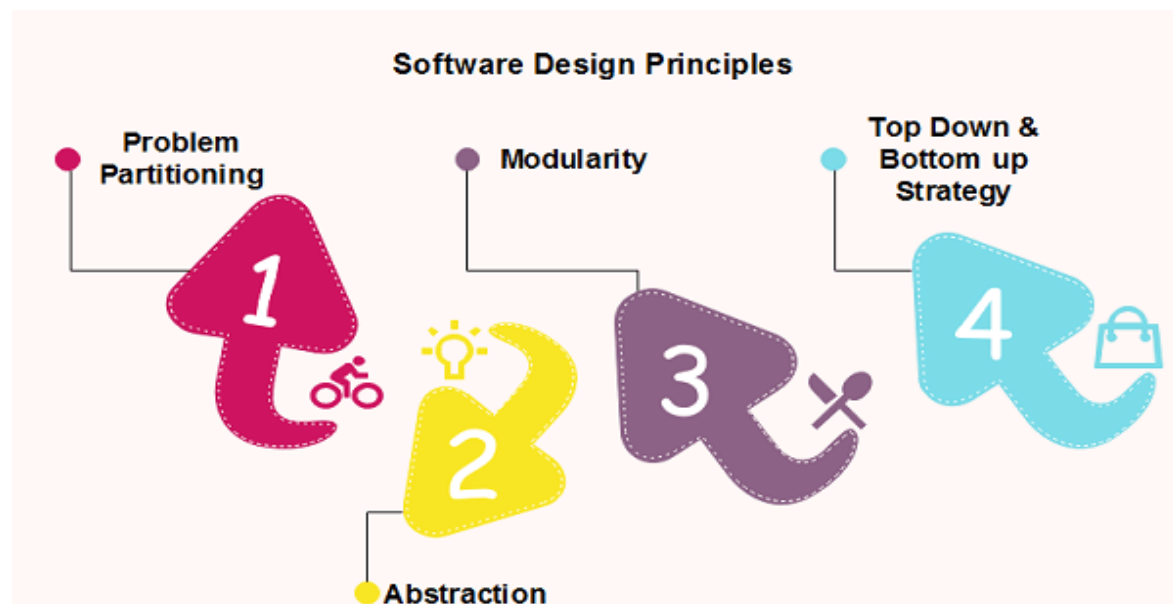
The design should be reviewed which means that the overall evaluation should be done to check if there is any error present or if it can be minimized.

10. Design is not coding and coding is not design –

Design means describing the logic of the program to solve any problem and coding is a type of language that is used for the implementation of a design.

Software design principles are concerned with providing means to handle the complexity of the design process effectively. Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.

2. Describe the Software Design Concepts



Concepts are defined as a principal idea or invention that comes into our mind or in thought to understand something. 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, the logic, or

thinking behind how you will design software. It allows the software engineer to create the model of the system or 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.

i) Problem Partitioning

For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

For software design, the goal is to divide the problem into manageable pieces.

Benefits of Problem Partitioning

1. Software is easy to understand
2. Software becomes simple
3. Software is easy to test
4. Software is easy to modify
5. Software is easy to maintain
6. Software is easy to expand

These pieces cannot be entirely independent of each other as they together form the system. They have to cooperate and communicate to solve the problem. This communication adds complexity.

Note: As the number of partition increases = Cost of partition and complexity increases

ii) Abstraction

An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation.

Abstraction can be used for existing element as well as the component being designed.

Here, there are two common abstraction mechanisms

1. Functional Abstraction
2. Data Abstraction

Functional Abstraction

- i. A module is specified by the method it performs.
- ii. The details of the algorithm to accomplish the functions are not visible to the user of the function.

Functional abstraction forms the basis for **Function oriented design approaches**.

Data Abstraction

Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for **Object Oriented design approaches**.

iii) Modularity

Modularity specifies to **the division of software into separate modules** which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable. Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

The desirable properties of a modular system are:

- Each module is a well-defined system that can be used with other applications.
- Each module has single specified objectives.
- Modules can be separately compiled and saved in the library.
- Modules should be easier to use than to build.

- Modules are simpler from outside than inside.

Advantages of Modularity

- It allows large programs to be written by several or different people
- It encourages the creation of commonly used routines to be placed in the library and used by other programs.
- It simplifies the overlay procedure of loading a large program into main storage.
- It provides more checkpoints to measure progress.
- It provides a framework for complete testing, more accessible to test
- It produced the well designed and more readable program.

Disadvantages of Modularity

- Execution time maybe, but not certainly, longer
- Storage size perhaps, but is not certainly, increased
- Compilation and loading time may be longer
- Inter-module communication problems may be increased
- More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done

Modular Design

Modular design reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system. We discuss a different section of modular design in detail in this section:

1. Functional Independence: Functional independence is achieved by developing functions that perform only one kind of task and do not excessively interact with other modules. Independence is important because it makes implementation more accessible and faster. The independent modules are easier to maintain, test, and reduce error propagation and can be reused in other programs as well. Thus, functional independence is a good design feature which ensures software quality.

It is measured using two criteria:

- **Cohesion:** It measures the relative function strength of a module.

- **Coupling:** It measures the relative interdependence among modules.

2. Information hiding: The fundamental of Information hiding suggests that modules can be characterized by the design decisions that protect from the others, i.e., In other words, modules should be specified that data include within a module is inaccessible to other modules that do not need for such information.

The use of information hiding as design criteria for modular system provides the most significant benefits when modifications are required during testing's and later during software maintenance. This is because as most data and procedures are hidden from other parts of the software, inadvertent errors introduced during modifications are less likely to propagate to different locations within the software.

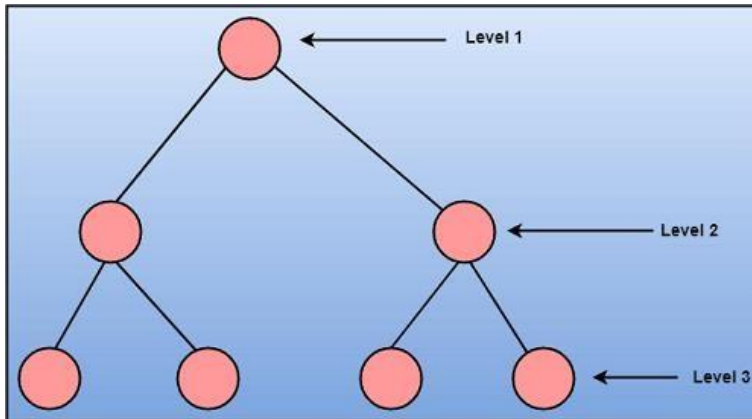
iv) Strategy of Design

A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.

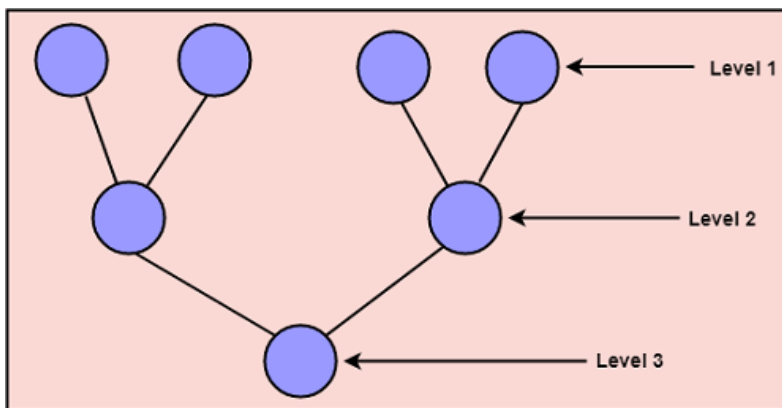
To design a system, there are two possible approaches:

1. Top-down Approach
2. Bottom-up Approach

1. Top-down Approach: This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.



2. Bottom-up Approach: A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.



Software design is a process to conceptualize the software requirements into software implementation. Software design takes the user requirements as challenges and tries to find optimum solution. While the software is being conceptualized, a plan is chalked out to find the best possible design for implementing the intended solution.

1. Abstraction

- A solution is stated in large terms using the language of the problem environment at the highest level abstraction.
- The lower level of abstraction provides a more detail description of the solution.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.

- A collection of data that describes a data object is a data abstraction.

2. Architecture

- The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.

3. Patterns

- A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

4. Modularity

- A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.

5. Information hiding

- Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

6. Functional independence

- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- The functional independence is accessed using two criteria i.e Cohesion and coupling.

Cohesion

- Cohesion is an extension of the information hiding concept.
- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

Coupling

- Coupling is an indication of interconnection between modules in a structure of software.

7. Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

8. Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- Refactoring is the process of changing the software system in a way that it does not change the external behaviour of the code still improves its internal structure.

9. Design classes

- The model of software is defined as a set of design classes.
- Every class describes the elements of problem domain and that focus on features of the problem which are user visible

3. Describe Structured Design

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. Benefit of structured design is, it gives better understanding of how the problem is being solved. Structured design also makes it simpler for designer to concentrate on the problem more accurately.

Structured design is mostly based on '**divide and conquer**' strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

The small pieces of problem are solved by means of solution modules. Structured design emphasis that these modules be well organized in order to achieve precise solution.

These modules are arranged in **hierarchy**. They communicate with each other. A good structured design always follows some rules for communication among multiple modules, namely -

Cohesion - grouping of all functionally related elements.

Coupling - communication between different modules.

A good structured design has high cohesion and low coupling arrangements.

4. Describe Functional Oriented Design

In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.

Function oriented design inherits some properties of structured design where divide and conquer methodology is used.

This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation.. These functional modules can share information among themselves by means of information passing and using information available globally.

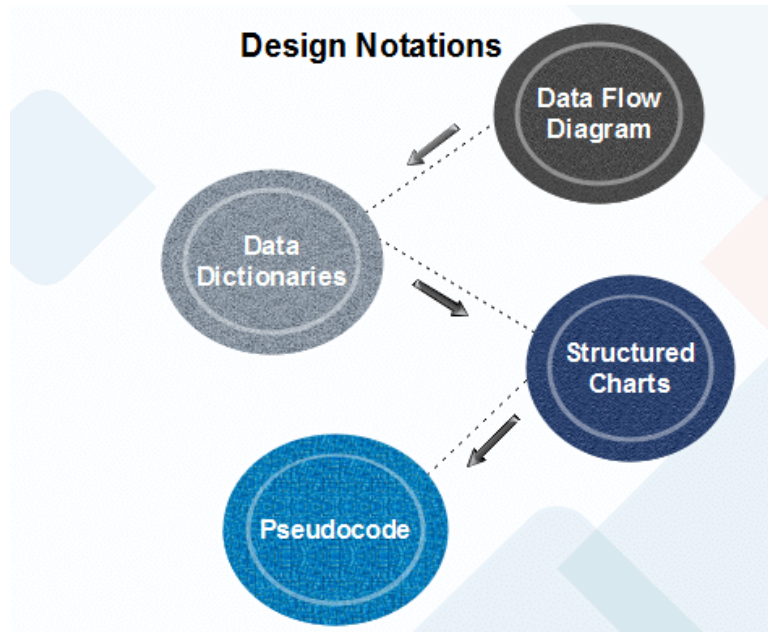
Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptable by other modules. Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

Design Process

- The whole system is seen as how data flows in the system by means of data flow diagram.
- DFD depicts how changes data and state of entire system functions.
- The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.
- Each function is then described at large.

Design Notations





Design Notations are primarily meant to be used during the process of design and are used to [represent design or design decisions](#). For a function-oriented design, the design can be represented graphically or mathematically by the following:



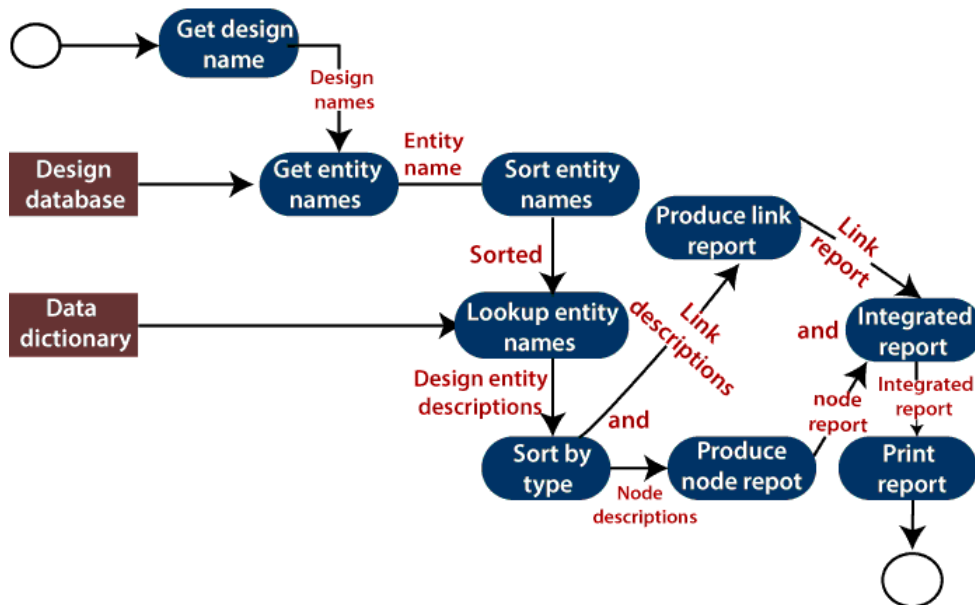
Data Flow Diagram

Data-flow design is concerned with designing a series of functional transformations that convert system inputs into the required outputs. The design is described as data-flow diagrams. These diagrams show how data flows through a system and how the output is derived from the input through a series of functional transformations.

The notation which is used is based on the following symbols:

| Symbol | Name | Meaning |
|---|-------------------|--|
|  | Rounded Rectangle | It represents functions which transforms input to output. The transformation name indicates its function. |
|  | Rectangle | It represents data stores. Again, they should give a descriptive name. |
|  | Circle | It represents user interactions with the system that provides input or receives output. |
|  | Arrows | It shows the direction of data flow. Their name describes the data flowing along the path. |
| "and" and "or" | Keywords | The keywords "and" and "or". These have their usual meanings in boolean expressions. They are used to link data flows when more than one data flow may be input or output from a transformation. |

Data flow diagram of a design report generator



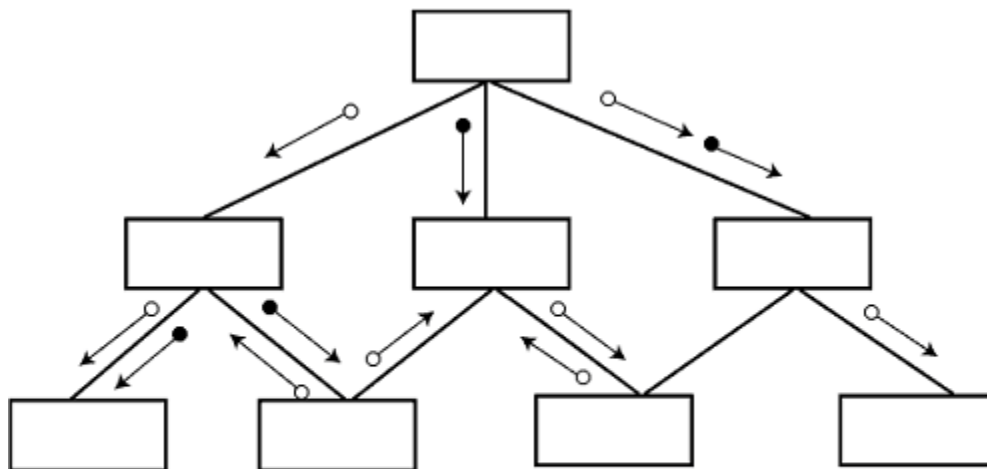
Data Dictionaries

A data dictionary lists all data elements appearing in the DFD model of a system. The data items listed contain all data flows and the contents of all data stores looking on the DFDs in the DFD model of a system.

- The data dictionary provides the analyst with a means to determine the definition of various data structures in terms of their component elements.

Structured Charts

It partitions a system into block boxes. A Black box system that functionality is known to the user without the knowledge of internal design.


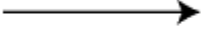


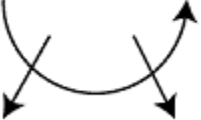
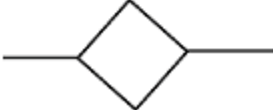


Hierarchical format of a structure chart

Structured Chart is a graphical representation which shows:

- System partitions into modules
- Hierarchy of component modules
- The relation between processing modules
- Interaction between modules
- Information passed between modules

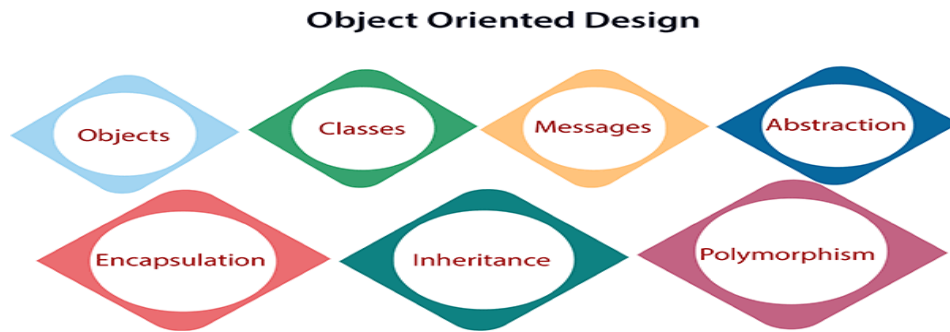
The following notations are used in structured chart:

| SYMBOL | DESCRIPTION |
|--|--------------|
|  | Module |
|  | Arrow |
|  | Data couple |
|  | Control Flag |
|  | Loop |
|  | Decision |

Pseudo-code

Pseudo-code notations can be used in both the preliminary and detailed design phases. Using pseudo-code, the designer describes system characteristics using short, concise, English Language phrases that are structured by keywords such as If-Then-Else, While-Do, and End.

5. Describe Object Oriented Design and OOD process



Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics. The whole concept of software solution revolves around the engaged entities.

Let us see the important concepts of Object Oriented Design:

- **Objects** - All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on the attributes.
- **Classes** - A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object.

In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.

- **Messages:** Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.
- **Abstraction** In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.

- **Encapsulation** - In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.
- **Inheritance** - OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.
- **Polymorphism** - OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

6. Describe the user interface design/system interface design

- User interface is the front-end application view to which user interacts in order to use the software.

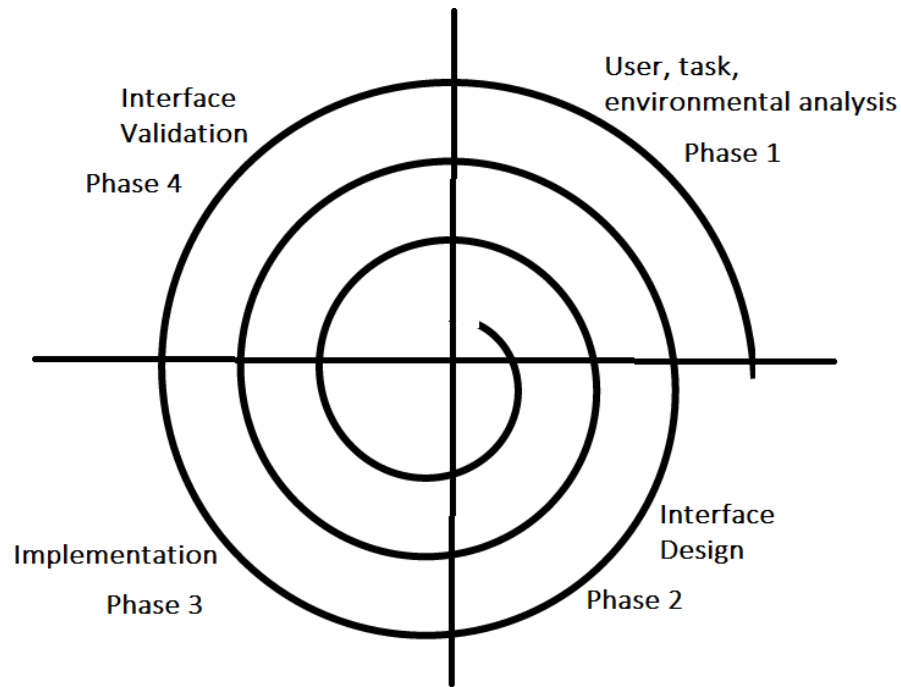
The software becomes more popular if its user interface is:

- 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.

User Interface Design Process:



The analysis and design process of a user interface is iterative and can be represented by a spiral model. The analysis and design process of user interface consists of four framework activities.

1. User, task, environmental analysis, and modelling:

Initially, the focus is based on the profile of users who will interact with the system, i.e. understanding, skill and knowledge, type of user, etc, based on the user's profile users are made into categories. From each category requirements are gathered. Based on the requirements developer understand how to develop the interface. Once all the requirements are gathered a detailed analysis is conducted. In the analysis part, the tasks that the user performs to establish the goals of the system are identified, described and elaborated. The analysis of the user environment focuses on the physical work environment. Among the questions to be asked are:

- Where will the interface be located physically?

- Will the user be sitting, standing, or performing other tasks unrelated to the interface?
 - Does the interface hardware accommodate space, light, or noise constraints?
 - Are there special human factors considerations driven by environmental factors?
2. **Interface Design:** The goal of this phase is **to define the set of interface objects and actions** i.e. **Control mechanisms that enable the user to perform desired tasks**. Indicate how these control mechanisms affect the system. Specify the action sequence of **tasks and subtasks**, also called a **user scenario**. Indicate the state of the system when the user performs a particular task. Always follow **the three golden rules stated by Theo Mandel**. **Design issues such as response time, command and action structure, error handling, and help facilities** are considered as the design model is refined. This phase serves as the foundation for the implementation phase.
 3. **Interface construction and implementation:** The implementation activity **begins with the creation of prototype (model)** that enables usage scenarios to be evaluated. As **iterative design process** continues a **User Interface toolkit** that allows **the creation of windows, menus, device interaction, error messages, commands, and many other elements of an interactive environment** can be used for completing the construction of an interface.
 4. **Interface Validation:** This phase focuses on testing the interface. The interface should be in such a way that **it should be able to perform tasks correctly and it should be able to handle a variety of tasks**. It should achieve all the user's requirements.

Golden Rules:

The following are the golden rules stated by Theo Mandel that must be followed during the design of the interface. **Place the user in control:**

- Define the interaction modes in such a way that **does not force the user into unnecessary or undesired actions**: The user should be able to easily enter and exit the mode with little or no effort.
- **Provide for flexible interaction**: Different people will **use different interaction mechanisms**, some might use keyboard commands, some might use mouse, some might use touch screen, etc, Hence all interaction mechanisms should be provided.
- **Allow user interaction to be interruptible and undoable**: When a user is doing a sequence of actions the user must be able to interrupt the sequence to do **some other work without losing the work** that had been done. **The user should also be able to do undo operation.**
- **Streamline interaction as skill level advances and allow the interaction to be customized**: **Advanced or highly skilled user** should be provided a chance to customize the interface as user wants which allows different interaction mechanisms so that **user doesn't feel bored** while using the same interaction mechanism.
- **Hide technical internals from casual users**: **The user should not be aware of the internal technical details of the system.** He should interact with the interface just to do his work.
- **Design for direct interaction with objects that appear on screen**: **The user should be able to use the objects and manipulate the objects** that are present on the screen to perform a necessary task. By this, the user feels easy to control over the screen.

Reduce the user's memory load:

- **Reduce demand on short-term memory**: When users are involved in some complex tasks the demand on short-term memory is significant. So the **interface should be designed in such a way to reduce the remembering of previously done actions**, given inputs and results.
- **Establish meaningful defaults**: Always initial set of defaults should be provided to the average user, **if a user needs to add some new features then he should be able to add the required features.**

- **Define shortcuts that are intuitive:** Mnemonics should be used by the user. Mnemonics means the keyboard shortcuts to do some action on the screen.
- **The visual layout of the interface should be based on a real-world metaphor:** Anything you represent on a screen if it is a metaphor for real-world entity then users would easily understand.
- **Disclose information in a progressive fashion:** The interface should be organized hierarchically i.e. on the main screen the information about the task, an object or some behavior should be presented first at a high level of abstraction. More detail should be presented after the user indicates interest with a mouse pick.

Make the interface consistent:

- **Allow the user to put the current task into a meaningful context:** Many interfaces have dozens of screens. So it is important to provide indicators consistently so that the user know about the doing work. The user should also know from which page has navigated to the current page and from the current page where can navigate.
- **Maintain consistency across a family of applications:** The development of some set of applications all should follow and implement the same design, rules so that consistency is maintained among applications.
- If past interactive models have created user expectations do not make changes unless there is a compelling reason.

7. Describe the design process.

Design Process

Software design process can be perceived as series of well-defined steps. Though it varies according to design approach (function oriented or object oriented, yet It may have the following steps involved:

- A solution design is created from requirement or previous used system and/or system sequence diagram.

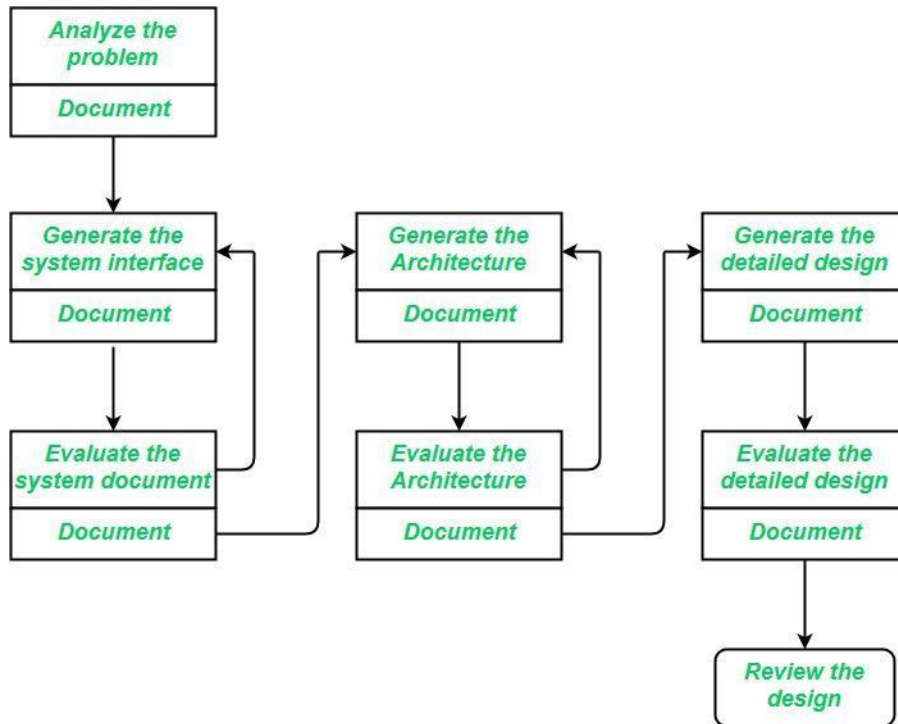
- Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
- Class hierarchy and relation among them is defined.
- Application framework is defined.

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

1. **Interface Design**
2. **Architectural Design**
3. **Detailed Design**

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.



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:

- Precise description of events in the environment, or messages from agents to which the system must respond.
- Precise description of the events or messages that the system must produce.
- Specification on the data, and the formats of the data coming into and going out of the system.
- 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:

- Gross decomposition of the systems into major components.
- Allocation of functional responsibilities to components.
- Component Interfaces
- Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
- 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:

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:

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

Software Design Approaches

Here are two generic approaches for software designing:

Top Down Design

We know that a system is composed of more than one sub-systems and it contains a number of components. Further, these sub-systems and components may have their own set of sub-system and components and creates hierarchical structure in the system.

Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved.

Top-down design starts with a generalized model of system and keeps on defining the more specific part of it. When all components are composed the whole system comes into existence.

Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.

Bottom-up Design

The bottom up design model starts with most specific and basic components. It proceeds with composing higher level of components by using basic or lower level components. It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction is increased.

Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.

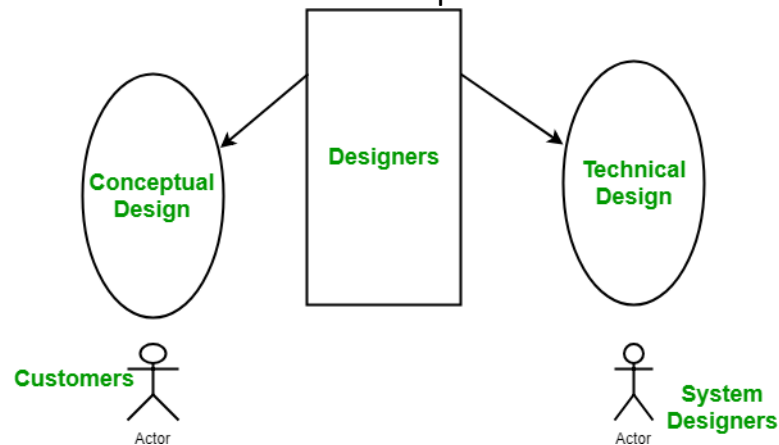
Both, top-down and bottom-up approaches are not practical individually. Instead, a good combination of both is used.

8. Briefly explain Coupling and Cohesion.

Introduction: The purpose of Design phase in the Software Development Life Cycle is to produce a solution to a problem given in the SRS(Software Requirement Specification) document. The output of the design phase is Software Design Document (SDD).

Basically, design is a two-part iterative process. First part is Conceptual Design that tells the customer what the system will do. Second is Technical Design that allows the system builders to understand the actual hardware and software

needed to solve customer's problem.



Conceptual design of the system:

- Written in simple language i.e. customer understandable language.
- Detailed explanation about system characteristics.
- Describes the functionality of the system.
- It is independent of implementation.
- Linked with requirement document.

Technical Design of the system:

- Hardware component and design.
- Functionality and hierarchy of software components.
- Software architecture

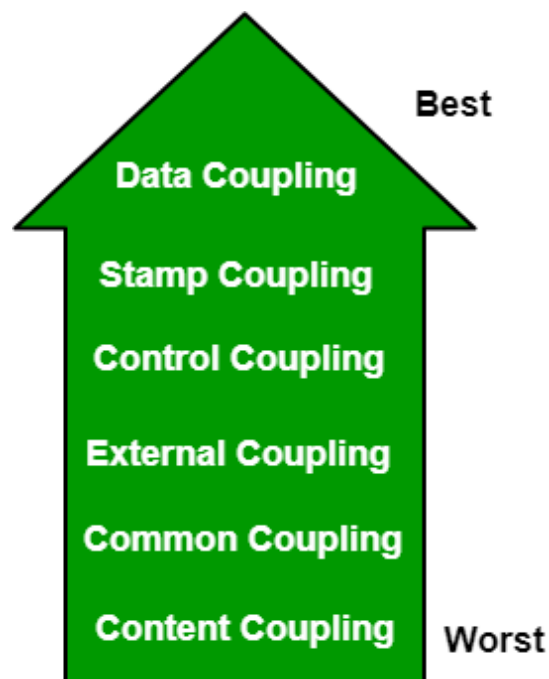
- Network architecture
- Data structure and flow of data.
- I/O component of the system.
- Shows interface.

Modularization: Modularization is the process of dividing a software system into multiple independent modules where each module works independently. There are many advantages of Modularization in software engineering. Some of these are given below:

- Easy to understand the system.
- System maintenance is easy.
- A module can be used many times as their requirements. No need to write it again and again.

9. Explain various types of Coupling.

Coupling: Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.



Types of Coupling:

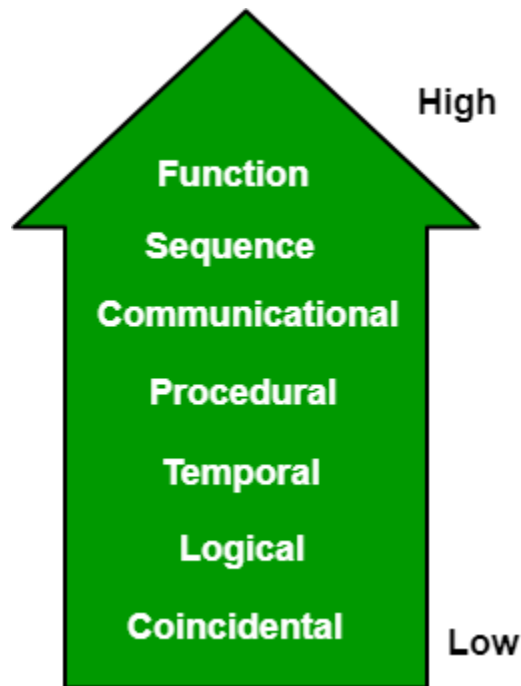
Data Coupling: If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data. Module communications don't contain tramp data.

Example-customer billing system.

- **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice was made by the insightful designer, not a lazy programmer.
- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
- **Common Coupling:** The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses, and reduced maintainability.
- **Content Coupling:** In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

10.Explain different types of Cohesion.

Cohesion: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



Types of Cohesion:

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.
- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for

these functions is in the same component. Operations are related, but the functions are significantly different.

- **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

11. Explain different phases of system design process with a neat diagram.

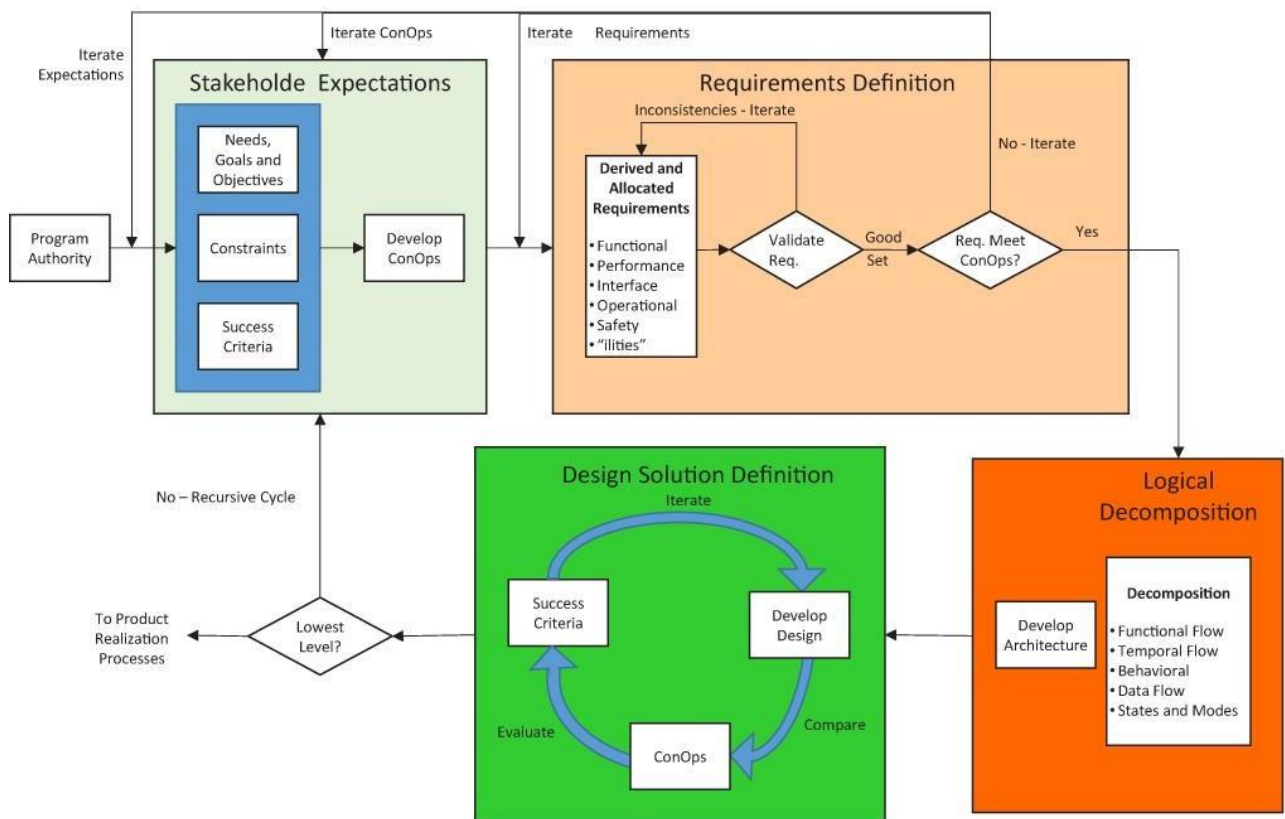
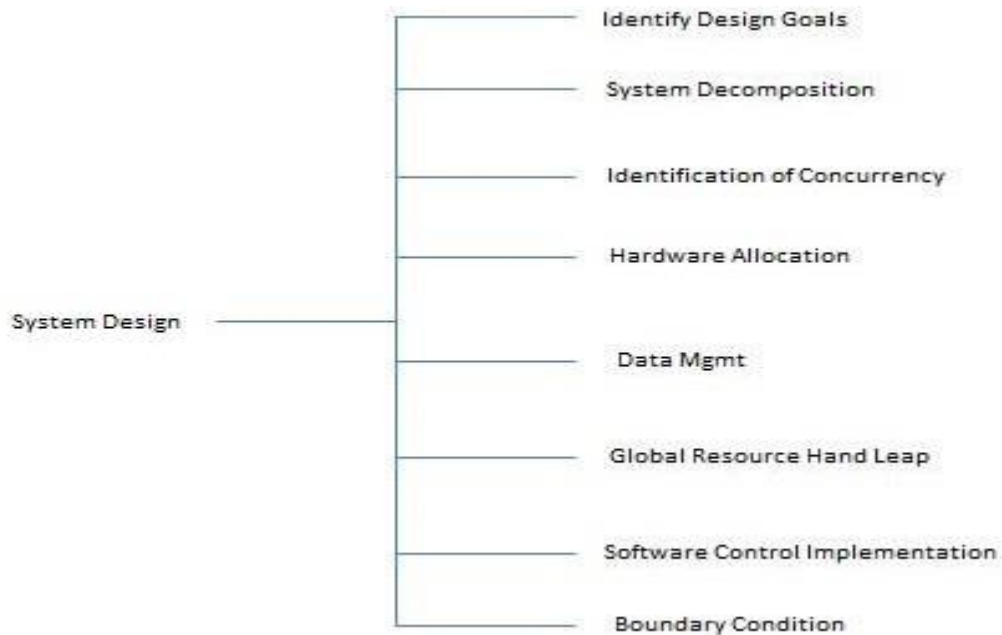


FIGURE 4.0-1 Interrelationships among the System Design Processes

system design is the phase that bridges the gap between problem domain and the existing system in a manageable way. This phase focuses on the solution domain, i.e. "how to implement?"

It is the phase where the SRS document is converted into a format that can be implemented and decides how the system will operate.

In this phase, the complex activity of system development is divided into several smaller sub-activities, which coordinate with each other to achieve the main objective of system development.



Inputs to System Design

System design takes the following inputs –

- Statement of work
- Requirement determination plan
- Current situation analysis
- Proposed system requirements including a conceptual data model, modified DFDs, and Metadata (data about data).

Outputs for System Design

System design gives the following outputs –

- Infrastructure and organizational changes for the proposed system.
- A data schema, often a relational schema.
- Metadata to define the tables/files and columns/data-items.
- A function hierarchy diagram or web page map that graphically describes the program structure.

- Actual or pseudocode for each module in the program.
- A prototype for the proposed system.

Types of System Design

Logical Design

Logical design pertains to an abstract representation of the data flow, inputs, and outputs of the system. It describes the inputs (sources), outputs (destinations), databases (data stores), procedures (data flows) all in a format that meets the user requirements.

While preparing the logical design of a system, the system analyst specifies the user needs at level of detail that virtually determines the information flow into and out of the system and the required data sources. Data flow diagram, E-R diagram modeling are used.

Physical Design

Physical design relates to the actual input and output processes of the system. It focuses on how data is entered into a system, verified, processed, and displayed as output.

It produces the working system by defining the design specification that specifies exactly what the candidate system does. It is concerned with user interface design, process design, and data design.

It consists of the following steps –

- Specifying the input/output media, designing the database, and specifying backup procedures.
- Planning system implementation.
- Devising a test and implementation plan, and specifying any new hardware and software.
- Updating costs, benefits, conversion dates, and system constraints.

Architectural Design

It is also known as high level design that focuses on the design of system architecture. It describes the structure and behavior of the system. It defines the structure and relationship between various modules of system development process.

Detailed Design

It follows Architectural design and focuses on development of each module.

Conceptual Data Modeling

It is representation of organizational data which includes all the major entities and relationship. System analysts develop a conceptual data model for the current system that supports the scope and requirement for the proposed system.

The main aim of conceptual data modeling is to capture as much meaning of data as possible. Most organization today use conceptual data modeling using E-R model which uses special notation to represent as much meaning about data as possible.



Entity Relationship Model








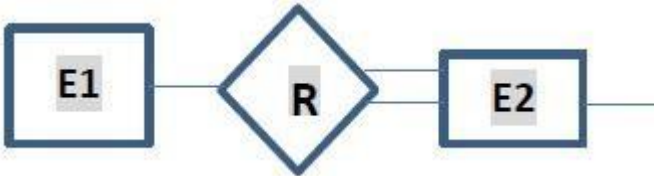
It is a technique used in database design that helps describe the relationship between various entities of an organization.

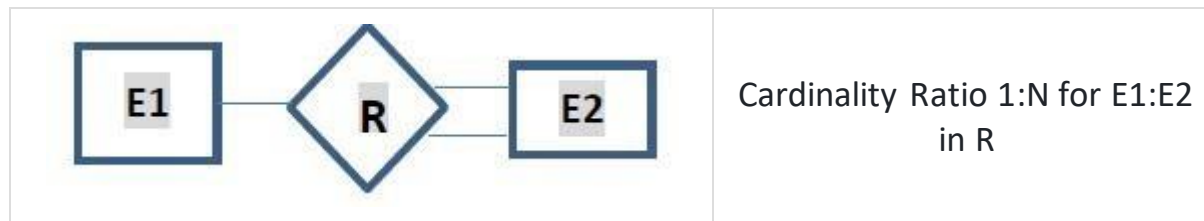
Terms used in E-R model

- **ENTITY** – It specifies distinct real world items in an application. For example: vendor, item, student, course, teachers, etc.
- **RELATIONSHIP** – They are the meaningful dependencies between entities. For example, vendor supplies items, teacher teaches courses, then supplies and course are relationship.
- **ATTRIBUTES** – It specifies the properties of relationships. For example, vendor code, student name. Symbols used in E-R model and their respective meanings –

The following table shows the symbols used in E-R model and their significance –

| Symbol | Meaning |
|---|-------------|
|  | Entity |
|  | Weak Entity |

| | |
|---|--------------------------------|
|  | Relationship |
|  | Identity Relationship |
|  | Attributes |
|  | Key Attributes |
|  | Multivalued |
|  | Composite Attribute |
|  | Derived Attributes |
|  | Total Participation of E2 in R |



Three types of relationships can exist between two sets of data: one-to-one, one-to-many, and many-to-many.

File Organization

It describes how records are stored within a file.

There are four file organization methods –

- **Serial** – Records are stored in chronological order (in order as they are input or occur). **Examples** – Recording of telephone charges, ATM transactions, Telephone queues.
- **Sequential** – Records are stored in order based on a key field which contains a value that uniquely identifies a record. **Examples** – Phone directories.
- **Direct (relative)** – Each record is stored based on a physical address or location on the device. Address is calculated from the value stored in the record's key field. Randomizing routine or hashing algorithm does the conversion.
- **Indexed** – Records can be processed both sequentially and non-sequentially using indexes.

Comparison

| | Serial | Sequential | Direct | Index |
|-----------------------------------|----------------------|--|--|-------------------------------|
| Type of Access | Batch | Batch | Online | Batch or Online |
| Data Organization | Serial | Sequentially by key value | No particular order | Sequentially and by index |
| Flexibility in handling inquiries | No | No | Yes | Yes |
| Availability of up to date Data | No | No | Yes | Yes |
| Speed Retrieval | Slow | Slow | Very Fast | Fast |
| Activity | High | High | Low | High |
| Volatility | Low | Low | High | High |
| Example | ATM Transition Queue | Payroll process script billing operation | Online reservation and banking transaction | Customer ordering and billing |

File Access

One can access a file using either Sequential Access or Random Access. File Access methods allow computer programs read or write records in a file.

Sequential Access

Every record on the file is processed starting with the first record until End of File (EOF) is reached. It is efficient when a large number of the records on the file need to be accessed at any given time. Data stored on a tape (sequential access) can be accessed only sequentially.

Direct (Random) Access

Records are located by knowing their physical locations or addresses on the device rather than their positions relative to other records. Data stored on a CD device (direct-access) can be accessed either sequentially or randomly.

Types of Files used in an Organization System

Following are the types of files used in an organization system –

- **Master file** – It contains the current information for a system. For example, customer file, student file, telephone directory.
- **Table file** – It is a type of master file that changes infrequently and stored in a tabular format. For example, storing Zipcode.
- **Transaction file** – It contains the day-to-day information generated from business activities. It is used to update or process the master file. For example, Addresses of the employees.
- **Temporary file** – It is created and used whenever needed by a system.
- **Mirror file** – They are the exact duplicates of other files. Help minimize the risk of downtime in cases when the original becomes unusable. They must be modified each time the original file is changed.
- **Log files** – They contain copies of master and transaction records in order to chronicle any changes that are made to the master file. It facilitates auditing and provides mechanism for recovery in case of system failure.
- **Archive files** – Backup files that contain historical versions of other files.

Documentation Control

Documentation is a process of recording the information for any reference or operational purpose. It helps users, managers, and IT staff, who require it. It is

important that prepared document must be updated on regular basis to trace the progress of the system easily.

After the implementation of system if the system is working improperly, then documentation helps the administrator to understand the flow of data in the system to correct the flaws and get the system working.

Programmers or systems analysts usually create program and system documentation. Systems analysts usually are responsible for preparing documentation to help users learn the system. In large companies, a technical support team that includes technical writers might assist in the preparation of user documentation and training materials.

Advantages

- It can reduce system downtime, cut costs, and speed up maintenance tasks.
- It provides the clear description of formal flow of present system and helps to understand the type of input data and how the output can be produced.
- It provides effective and efficient way of communication between technical and nontechnical users about system.
- It facilitates the training of new user so that he can easily understand the flow of system.
- It helps the user to solve the problems such as troubleshooting and helps the manager to take better final decisions of the organization system.
- It provides better control to the internal or external working of the system.

Types of Documentations

When it comes to System Design, there are following four main documentations –

- Program documentation
- System documentation
- Operations documentation
- User documentation

Program Documentation

- It describes inputs, outputs, and processing logic for all the program modules.
- The program documentation process starts in the system analysis phase and continues during implementation.

- This documentation guides programmers, who construct modules that are well supported by internal and external comments and descriptions that can be understood and maintained easily.

Operations Documentation

Operations documentation contains all the information needed for processing and distributing online and printed output. Operations documentation should be clear, concise, and available online if possible.

It includes the following information –

- Program, systems analyst, programmer, and system identification.
- Scheduling information for printed output, such as report, execution frequency, and deadlines.
- Input files, their source, output files, and their destinations.
- E-mail and report distribution lists.
- Special forms required, including online forms.
- Error and informational messages to operators and restart procedures.
- Special instructions, such as security requirements.

User Documentation

It includes instructions and information to the users who will interact with the system. For example, user manuals, help guides, and tutorials. User documentation is valuable in training users and for reference purpose. It must be clear, understandable, and readily accessible to users at all levels.

The users, system owners, analysts, and programmers, all put combined efforts to develop a user's guide.

A user documentation should include –

- A system overview that clearly describes all major system features, capabilities, and limitations.
- Description of source document content, preparation, processing, and, samples.
- Overview of menu and data entry screen options, contents, and processing instructions.
- Examples of reports that are produced regularly or available at the user's request, including samples.
- Security and audit trail information.

- Explanation of responsibility for specific input, output, or processing requirements.
- Procedures for requesting changes and reporting problems.
- Examples of exceptions and error situations.
- Frequently asked questions (FAQs).
- Explanation of how to get help and procedures for updating the user manual.

System Documentation

System documentation serves as the technical specifications for the IS and how the objectives of the IS are accomplished. Users, managers and IS owners need never reference system documentation. System documentation provides the basis for understanding the technical aspects of the IS when modifications are made.

- It describes each program within the IS and the entire IS itself.
- It describes the system's functions, the way they are implemented, each program's purpose within the entire IS with respect to the order of execution, information passed to and from programs, and overall system flow.
- It includes data dictionary entries, data flow diagrams, object models, screen layouts, source documents, and the systems request that initiated the project.
- Most of the system documentation is prepared during the system analysis and system design phases.
- During systems implementation, an analyst must review system documentation to verify that it is complete, accurate, and up-to-date, and including any changes made during the implementation process.

A good system design is to organize the program modules in such a way that are easy to develop and change. Structured design techniques help developers to deal with the size and complexity of programs. Analysts create instructions for the developers about how code should be written and how pieces of code should fit together to form a program.

Importance :

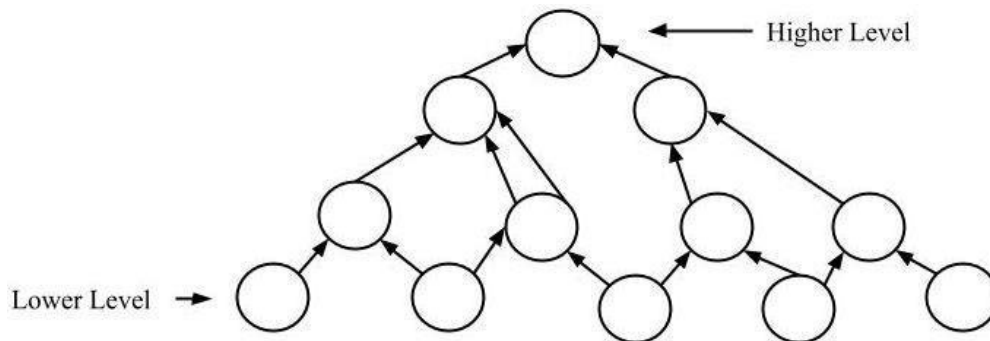
1. If any pre-existing code needs to be understood, organized, and pieced together.
2. It is common for the project team to have to write some code and produce original programs that support the application logic of the system.

There are many strategies or techniques for performing system design. They are:

- **Bottom-up approach:**

The design starts with the lowest level components and subsystems. By using these components, the next immediate higher-level components and subsystems are created or composed. The process is continued till all the components and subsystems are composed into a single component, which is considered as the complete system. The amount of abstraction grows high as the design moves to more high levels.

By using the basic information existing system, when a new system needs to be created, the bottom-up strategy suits the purpose.



Advantages:

- The economics can result when general solutions can be reused.
- It can be used to hide the low-level details of implementation and be merged with the top-down technique.

Disadvantages:

- It is not so closely related to the structure of the problem.
- High-quality bottom-up solutions are very hard to construct.
- It leads to the proliferation of 'potentially useful' functions rather than the most appropriate ones.

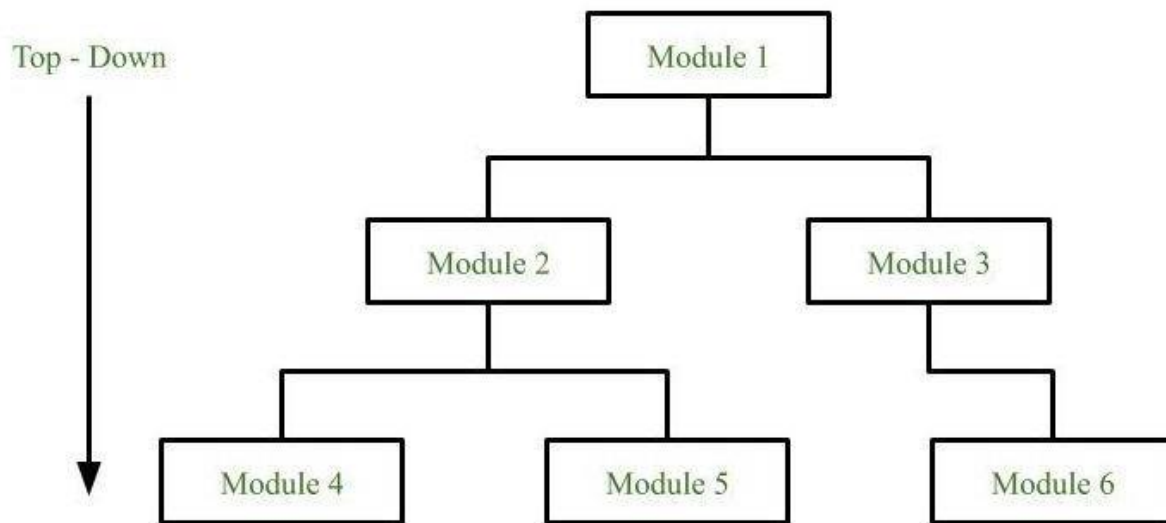
- **Top-down approach:**

Each system is divided into several subsystems and components. Each of the subsystems is further divided into a set of subsystems and components. This process of division facilitates in forming a system hierarchy structure. The

complete software system is considered as a single entity and in relation to the characteristics, the system is split into sub-system and component. The same is done with each of the sub-systems.

This process is continued until the lowest level of the system is reached. The design is started initially by defining the system as a whole and then keeps on adding definitions of the subsystems and components. When all the definitions are combined together, it turns out to be a complete system.

For the solutions of the software that need to be developed from the ground level, top-down design best suits the purpose.



Advantages:

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

Disadvantages:

- Project and system boundaries tend to be application specification-oriented. Thus it is more likely that advantages of component reuse will be missed.
- The system is likely to miss, the benefits of a well-structured, simple architecture.

- **Hybrid Design:**

It is a combination of both the top-down and bottom-up design strategies. In this, we can reuse the modules.

System Design in Software Development

System design is the process of designing the elements of a system such as the architecture, modules and components, the different interfaces of those components and the data that goes through that system.

System Analysis is the process that decomposes a system into its component pieces for the purpose of defining how well those components interact to accomplish the set requirements.

The purpose of the System Design process is to provide sufficient detailed data and information about the system and its system elements to enable the implementation consistent with architectural entities as defined in models and views of the system architecture.

Elements of a System

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

Major Tasks Performed During the System Design Process

1. Initialize design definition

- Plan for and Identify the technologies that will compose and implement the systems elements and their physical interfaces.
- Determine which technologies and system elements have a risk to become obsolete, or evolve during the operation stage of the system. Plan for their potential replacement.
- Document the design definition strategy, including the need for and requirements of any enabling systems, products, or services to perform the design.

2. Establish design characteristics

- Define the design characteristics relating to the architectural characteristics and check that they are implementable.
- Define the interfaces that were not defined by the System Architecture process or that need to be refined as the design details evolve.
- Define and document the design characteristics of each system element2.

3. Assess alternatives for obtaining system elements

- Assess the design options
- Select the most appropriate alternatives.
- If the decision is made to develop the system element, rest of the design definition process and the implementation process are used. If the decision is

to buy or reuse a system element, the acquisition process may be used to obtain the system element.

4. Manage the design

- Capture and maintain the rationale for all selections among alternatives and decisions for the design, architecture characteristics.
- Assess and control the evolution of the design characteristics.

Factors that Affect Technology Trade-offs during System Design

Scale of Product

- For example, enterprise software companies that are building system-level software prioritize reliability because customers need to use them. Each change needs to be rigorously tested, and often approved before it can be released.
- Meanwhile, consumer internet companies spend time and money on making their UX delightful so that people want to use them. Reliability is something they're willing to sacrifice. Since many are web-based applications, they can iterate quickly and release changes frequently.

Time

- Learning new technologies sometimes often takes time. The trade-offs in this instance will be made according to which stack/technology will be in time with the set delivery dates. If switching to a new stack/technology will result in a

major shift on the delivery dates and major inconveniences to the stakeholders then the switch can be held off until an appropriate time.

Cost

- On a larger scale Technology decisions are made based on which is more cost effective, where a comparison can be done on which will be more effective between buying an off the shelf system and customizing it or building a new system.

Efficiency

- Technology trade offs are also done based on which technology is more efficient for example choosing between ReactJs or AngularJs for a front end application.

User Experience and Support

- The amount of support and documentation available on a given technology can also be a determining factor on the decisions. Working with Technologies that have a large support base, comprehensive documentation and A good user experience is much easier and take a very short time to ramp up on due to the large amount of resources available to support it.

Maintainability

- maintainability in this case is the ease with which a product can be maintained in order to correct errors, fix bugs and add additional features. Trade-offs decisions will be made based on the maintainability of the Technology

Reliability

- In this case the trade offs are made based on the Technology that performs consistently well and consistently upgrading to more efficient versions.

Scalability

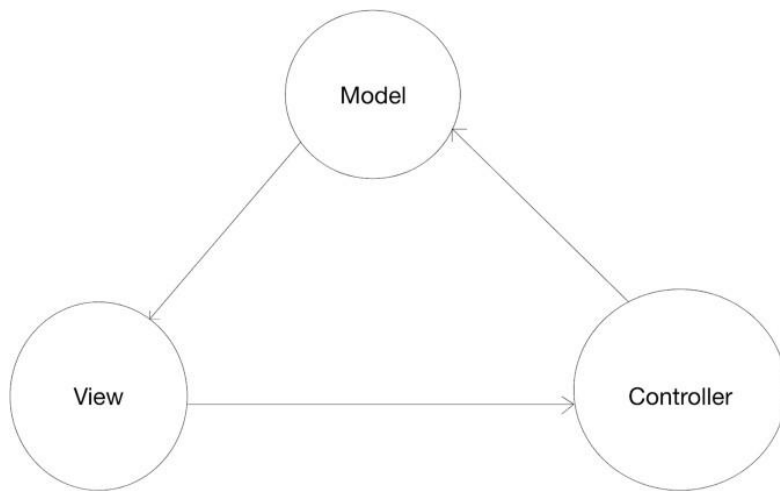
- Technology trade offs are also made based on the technologies that are more scalable and able to handle increase loads efficiently without a break in the system efficiency.

MVC Design pattern

The Model View Controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information.

MVC mostly relates to the user Interface/interaction layer of an application.

In the MVC pattern the user sees the **View** which is updated by the **model** which is turn manipulated by the **Controller**.



MVC Pattern

- The **Model** contains only the pure application data, it contains no logic describing how to present the data to a user. They are the parts of the application that implement the logic for the application's data domain. They retrieve and store model state in a database.
- The **View** presents the model's data to the user. The view can only be used to access the model's data. They are the components that display the application's user interface (UI).
- The **Controller** exists between the view and the model. It listens to events triggered by the view and executes the appropriate commands. They are the components that handle user interaction, work with the model, and ultimately select a view to render that displays UI.

Advantages of the MVC design pattern

- Multiple developers can work simultaneously on the model, controller and views.

- MVC enables logical grouping of related actions on a controller together. The views for a specific model are also grouped together.
- Low coupling — The very nature of the MVC framework is such that there is low coupling among models, views or controllers.
- Models can have multiple views.
- Ease of modification — Because of the separation of responsibilities, future development or modification is easier

Disadvantages

- Knowledge on multiple technologies becomes the norm. Developers using MVC need to be skilled in multiple technologies.

Below is an example of a System Design

