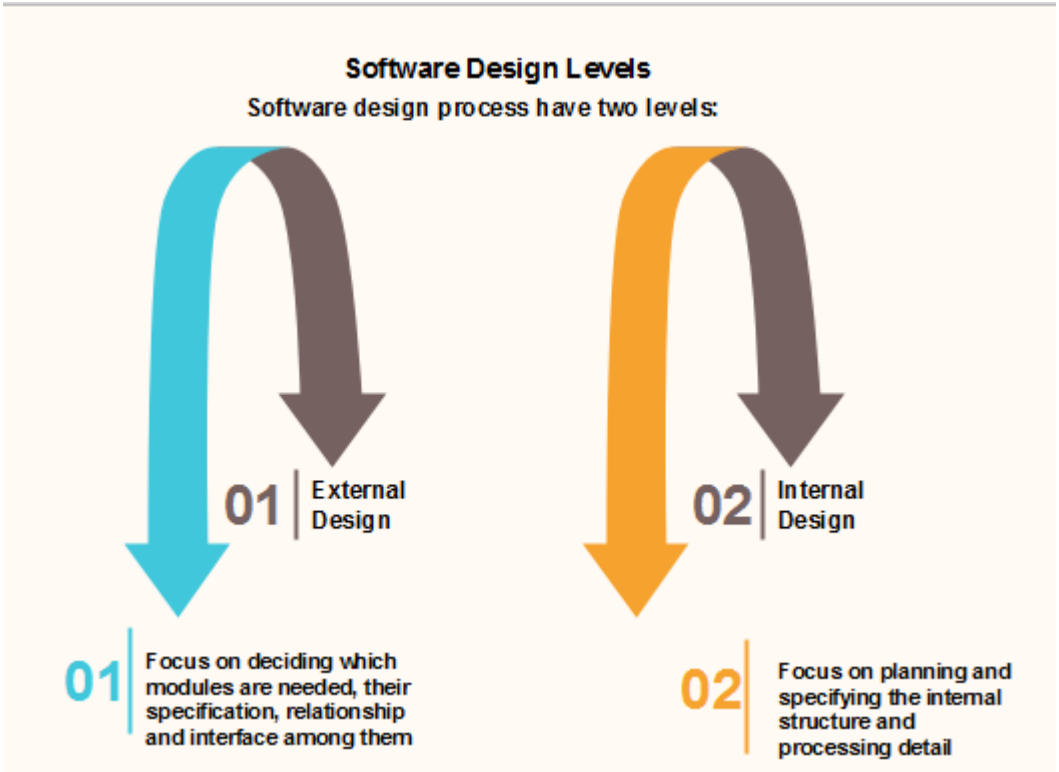
UNIT 4:

Software Design

Software Design

Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. It deals with representing the client's requirement, as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable using programming language.

The software design phase is the first step in **SDLC (Software Design Life Cycle)**, which moves the concentration from the problem domain to the solution domain. In software design, we consider the system to be a set of components or modules with clearly defined behaviors & boundaries.



Objectives of Software Design

Following are the purposes of Software design:

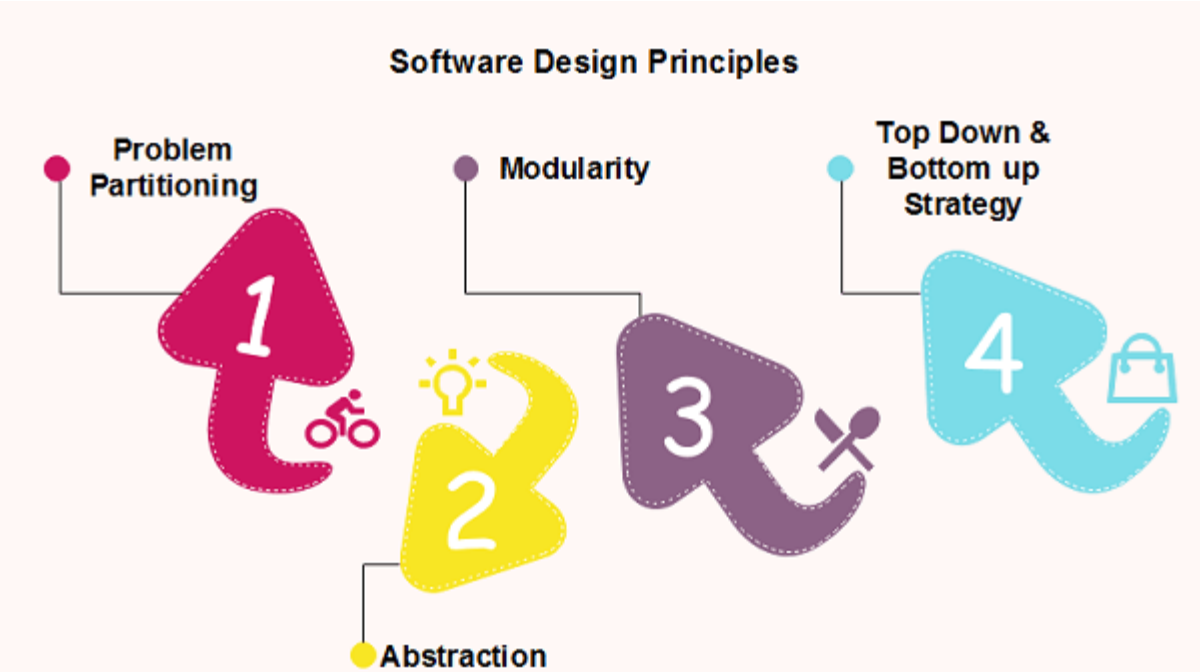
Objectives of Software Design

1. **Correctness:**Software design should be correct as per requirement.
2. **Completeness:**The design should have all components like data structures, modules, and external interfaces, etc.
3. **Efficiency:**Resources should be used efficiently by the program.
4. **Flexibility:**Able to modify on changing needs.
5. **Consistency:**There should not be any inconsistency in the design.
6. **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.

Software Design Principles

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.

Following are the principles of Software Design



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***

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**.

Modularity

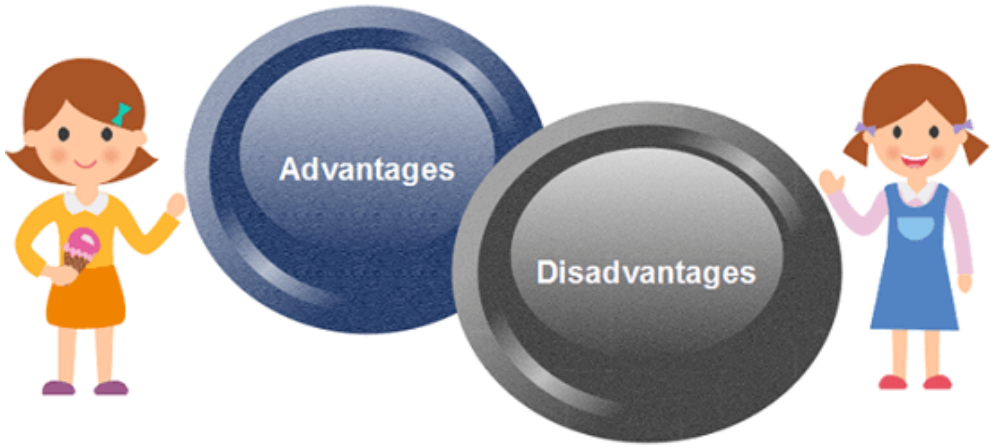
Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable. Single large programs are difficult 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 and Disadvantages of Modularity

In this topic, we will discuss various advantage and disadvantage of Modularity.



Advantages of Modularity

There are several 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

There are several 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.

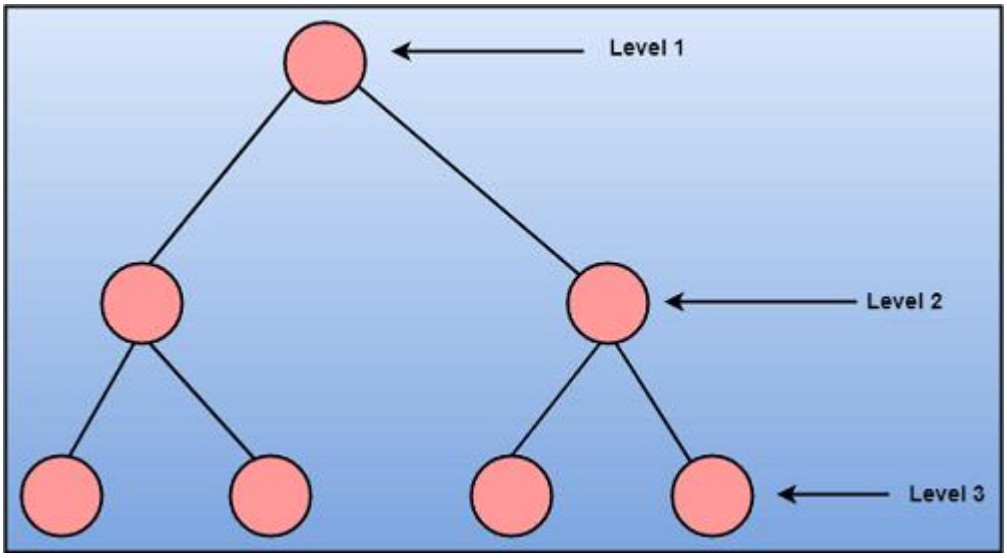
Strategy of Design

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.

What Is Design Methodology?

Design methodology refers to the development of a system or method for a unique situation. Today, the term is most often applied to technological fields in reference to web design, software or information systems design. Various degree programs involve design methodology, including those in the graphic and digital arts. Read this article to learn more about what design methodology is.

Data Design

Data design is the first design activity, which results in less complex, modular and efficient program structure. The information domain model developed during analysis phase is transformed into data structures needed for implementing the software. The data objects, attributes, and relationships depicted in entity relationship diagrams and the information stored in data dictionary provide a base for data design activity. During the data design process, data types are specified along with the integrity rules required for the data. For specifying and designing efficient data structures, some principles should be followed. These principles are listed below.

1. The data structures needed for implementing the software as well-as the operations that can be applied on them should be identified.
2. A data dictionary should be developed to depict how different data objects interact with each other and what constraints are to be imposed on the elements of data structure.
3. Stepwise refinement should be used in data design process and detailed design decisions should be made later in the process.
4. Only those modules that need to access data stored in a data structure directly should be aware of the representation of the data structure.
5. A library containing the set of useful data structures along with the operations that can be performed on them should be maintained.
6. Language used for developing the system should support abstract data types.

The structure of data can be viewed at three levels, namely, *program* component level, application level, and business level. At the **program component level,** the design of data structures and the algorithms required to manipulate them is necessary, if high-quality software is desired. At the **application level,** it is crucial to convert the data model into a database so that the specific business objectives of a system could be achieved. At the **business level,** the collection of information stored in different databases should be reorganized into data warehouse, which enables data mining that has an influential impact on the business.

Architectural Design

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.

The use of architectural styles is to establish a structure for all the components of the system.

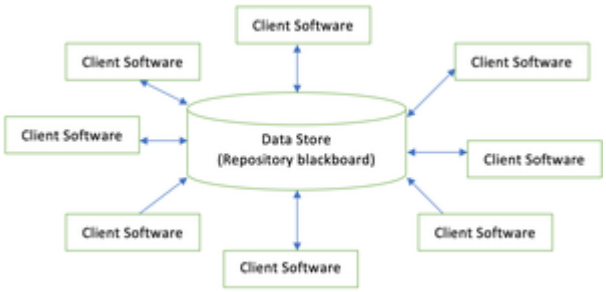
Taxonomy of Architectural styles:

1] Data centered 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.

Advantage of Data centered architecture

- Repository of data is independent of clients
- Client work independent of each other
- It may be simple to add additional clients.
- Modification can be very easy



Data centered architecture

2] Data flow architectures:

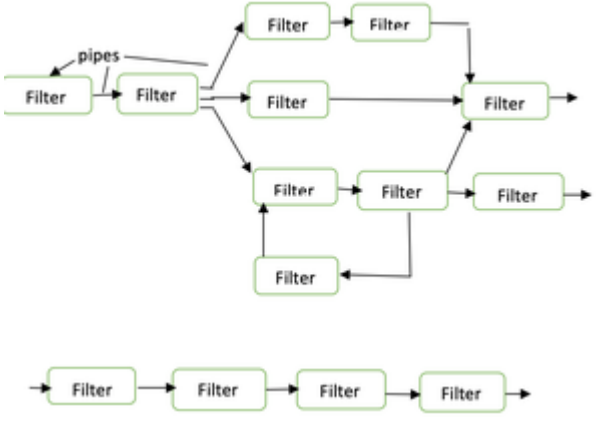
- This kind of architecture is used when input data is transformed into output data through a series of computational manipulative components.
- The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by lines.
- Pipes 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.

Advantages of Data Flow architecture

- It encourages upkeep, repurposing, and modification.
- With this design, concurrent execution is supported.

The disadvantage of Data Flow architecture

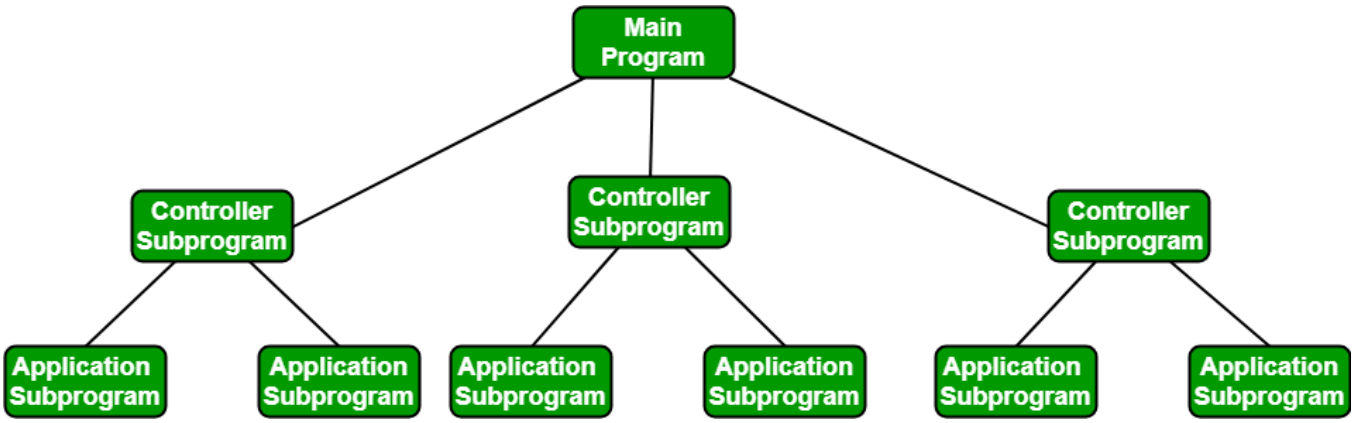
- It frequently degenerates to batch sequential system
- Data flow architecture does not allow applications that require greater user engagement.
- It is not easy to coordinate two different but related streams



Data Flow architecture

3] 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.

- **Remote procedure call architecture:** This components is used to present in a main program or sub program architecture distributed among multiple computers on a network.
- **Main program or Subprogram architectures:** The main program structure decomposes into number of subprograms or function into a control hierarchy. Main program contains number of subprograms that can invoke other components.



4] 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.

Characteristics of Object Oriented architecture

- Object protect the system's integrity.
- An object is unaware of the depiction of other items.

Advantage of Object Oriented architecture

- It enables the designer to separate a challenge into a collection of autonomous objects.
- Other objects are aware of the implementation details of the object, allowing changes to be made without having an impact on other objects.

5] 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.

Layered architecture:

Procedural Design

The objective in procedural design is to transform structural components into a procedural description of the software.

The step occurs after the data and program structures have been established, i.e. after architectural design. Procedural details can be represented in different ways:

1. Graphical Design Notation: The most widely used notation is the flowchart. Some notation used in flowcharts are
(i) Boxes to indicate processing steps.
(ii) Diamond to indicate logical condition.
(iii) Arrows to indicate flow of control.
(iv) Two boxes connected by a line of control will indicate a Sequence.

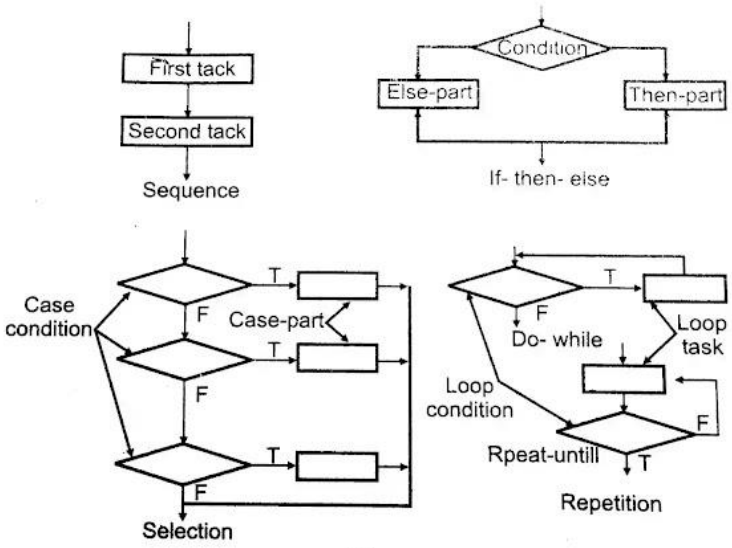


Fig.

2. Tabular Design Notation: (i) Decision tables provide a notation that translates actions and conditions (described a processing narrative) into a tabular form.

Conditions	Fixed rate account	1	2	3	4	5
	Variable rate	T	T	F	F	F
	Consumption < 100K WH	T	F	T	F	
	Consumption≥ 100 K WH	F	T	F	T	
Actions	Minimum monthly charge	X				
	Schedule A billing		X	X		
	Schedule B billing Other treatment				X	
	Other treatment					X

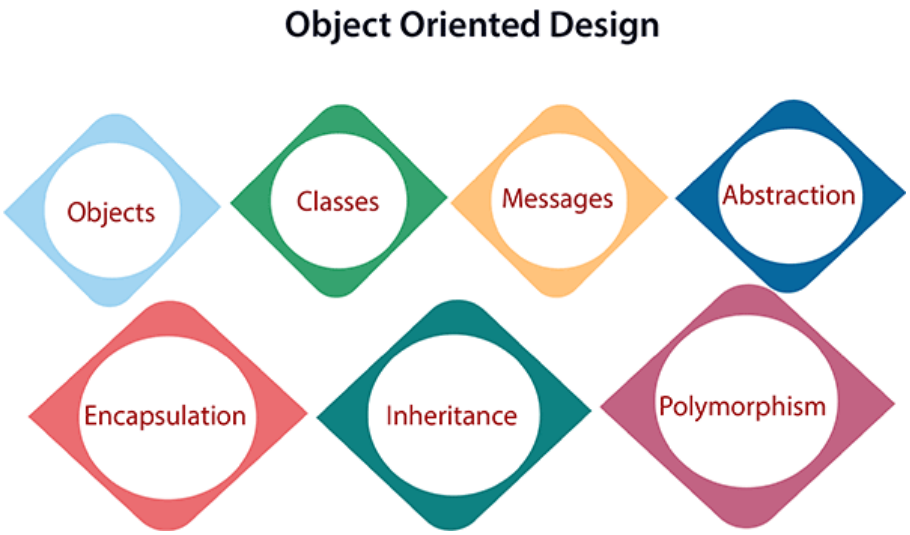
(ii) The upper left-hard section contains a list of all conditions. The lower left-hand section lists all actions that are possible based on the conditions. The right-hand sections form a matrix that indicates condition combinations and the corresponding actions that will occur for a specific combination.

3. Program Design Language: It is a method designing and documenting methods and procedures in software. It is related to pseudocode, but unlike pseudocode, it is written in plain language without any terms that could suggest the use of any programming language or library.

Object-Oriented Design

In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data. For example, in a Library Automation Software, each library representative may be a separate object with its data and functions to operate on these data. The tasks defined for one purpose cannot refer or change data of other objects. Objects have their internal data which represent their state. Similar objects create a class. In other words, each object is a member of some class. Classes may inherit features from the superclass.

The different terms related to object design are:



1. **Objects:** All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.
2. **Classes:** A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.
3. **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.
4. **Abstraction** In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.

5. **Encapsulation:** Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.
6. **Inheritance:** OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate superclasses.This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.
7. **Polymorphism:** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.