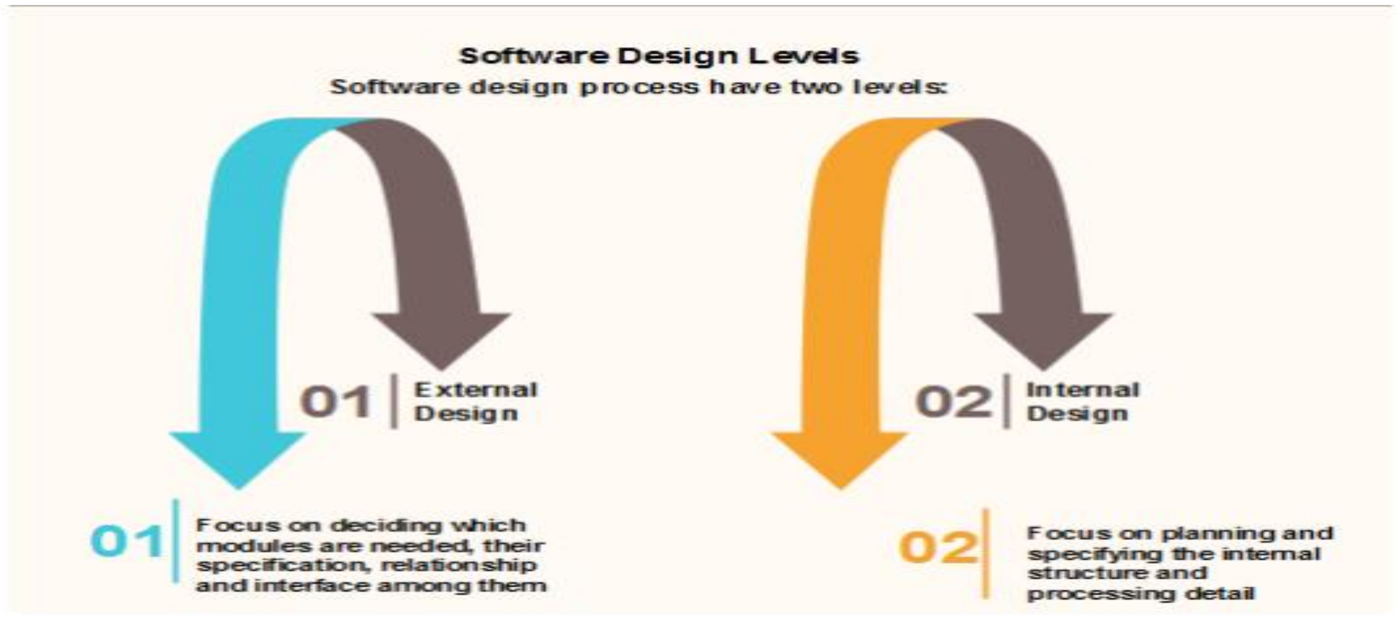
Design fundamental: Process, Data and Behavioural modelling, Design Concepts, Modularity, Architectural design, Coupling and Cohesion.

Software Design:

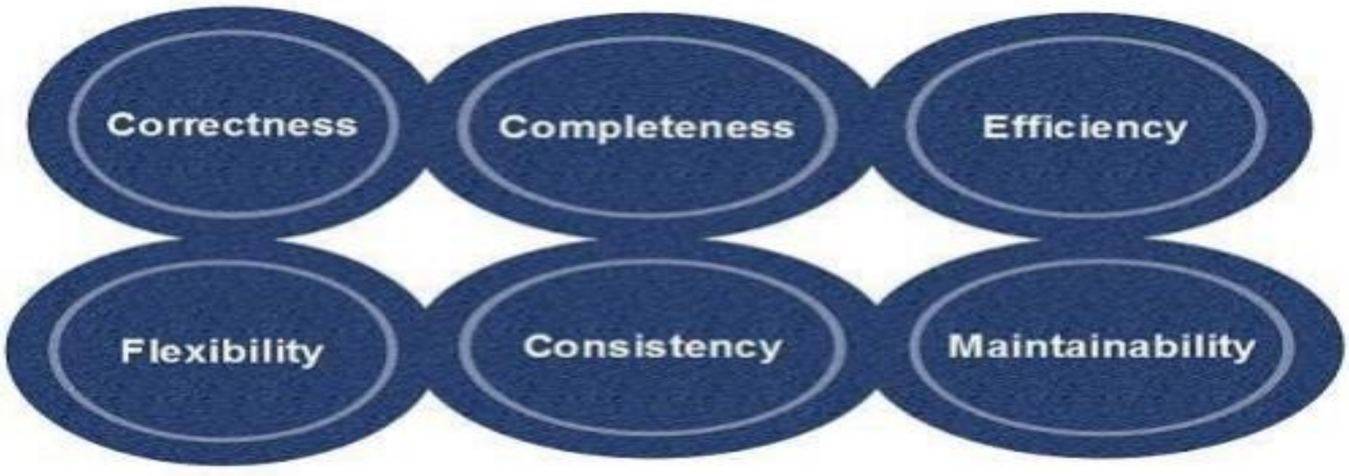
Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. It deals with representing the client's requirement, as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable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


Software Design Levels
Software design process have two levels:
01 | External Design
02 | Internal Design
01 | Focus on deciding which modules are needed, their specification, relationship and interface among them
02 | Focus on planning and specifying the internal structure and processing detail

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 Process (Different levels of Software Design)

• 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 or phases of design

1. Interface Design

2. Architectural Design

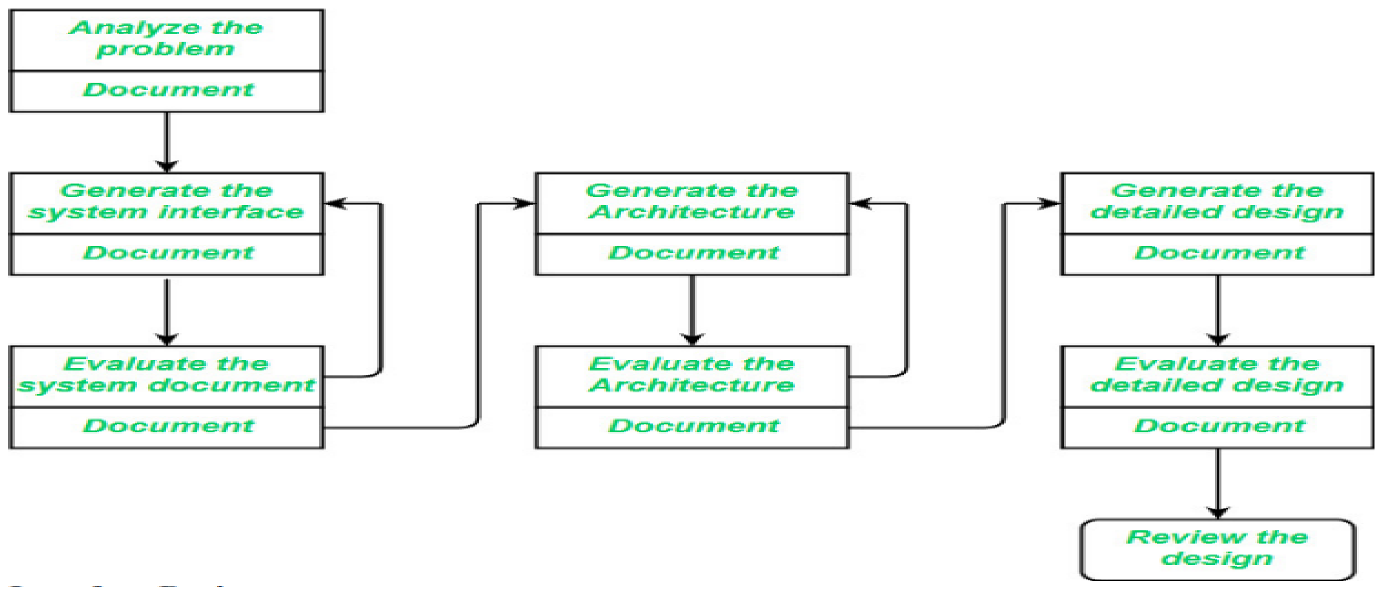
3. Detailed Design Elements of a System

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.



Analyze the problem
Document
Generate the system interface
Document
Evaluate the system document
Document
Generate the Architecture
Document
Evaluate the Architecture
Document
Generate the detailed design
Document
Evaluate the detailed design
Document
Review the design

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:

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.

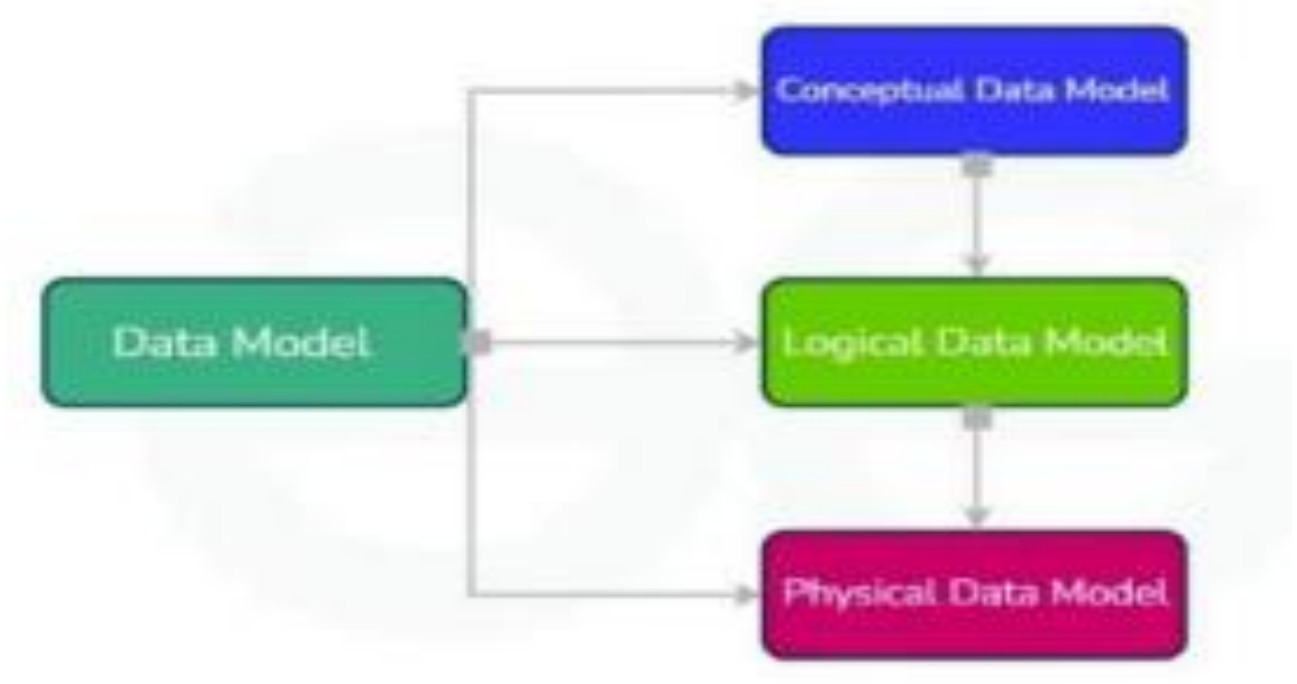
7. Algorithms and data structures.

Data modelling :-

Data models are visual representations of an enterprise's data elements and the connections between them. Models assist to define and arrange data in the context of key business processes, hence facilitating the creation of successful information systems. They let business and technical personnel to collaborate on how data will be kept, accessed, shared, updated, and utilised within an organisation.

Types of Data Models

There are three main types of data models:



• Conceptual Data Model: Conceptual Data Model is representations of data Examine and describe in depth your abstract, high-level business concepts and structures. They are most commonly employed when working through high-level concepts and preliminary needs at the start of a new project. They are typically developed as alternatives or preludes to the logical data models that come later, he main purpose of this data model is to organize, define business problems, rules and concepts. For instance, it helps business people to view any data like market data, customer data, and purchase data

• Logical Data Model: In the logical data model, by offering a thorough representation of the data at a logical level, the logical data model expands on the conceptual model. It outlines the tables, columns, connections, and constraints that make up the data structure. Although logical data models are not dependant on any particular database management system (DBMS), they are more similar to how data would be implemented in a database. The physical design of databases is based on this idea.

• Physical Data Model: In Physical Data model, the implementation is explained with reference to a particular database system. It outlines every part and service needed to construct a database. It is made with queries and the database language. Every table, column, and constraint—such as primary key, foreign key, NOT NULL, etc.—is represented in the physical data model. The creation of a database is the primary task of the physical data model. Developers and database administrators (DBAs) designed this model. This kind of data modelling aids in the creation of the schema and provides us with an abstraction of the databases. This model explains how the data model is specifically implemented. Constraints, RDBMS features, and database column keys are made possible by the physical data model.

Data Modelling Process

The practice of conceptually representing data items and their connections to one another is known as data modelling. Data modellers collaborate with stakeholders at each stage of the process to define entities and attributes, establish relationships between data objects, and create models that accurately represent the data in a format that can be consumed by applications. These stakeholders may include developers, database administrators, and other interested parties. Let’s discuss the data modelling steps:

1. Identifying data sources: The first stage is to identify and investigate the different sources of data both inside and outside the company. It’s critical to comprehend the sources of the data and how various sources add to the information as a whole. Determining the sources of data is essential since it guarantees a thorough framework for data modelling. It assists in gathering all pertinent data, setting the stage for a precise and comprehensive depiction of the data landscape.

2. Defining Entities and Attributes: This stage is all on identifying the entities (items or ideas) and the characteristics that go along with them. Entities constitute the subject matter of the data, whereas attributes specify the particular qualities of each entity. The foundation of data modelling is the definition of entities and characteristics. It offers an orderly and transparent framework, which is necessary to comprehend the characteristics of the data and create a useful model.

3. Mapping Relationships: Relationships show the connections or associations between various things. Relationship mapping entails locating and characterising these linkages, indicating the nature and cardinality of every relationship. In order to capture the interdependencies within the data, it is essential to understand relationships. It improves the correctness of the model by capturing the relationships between various data pieces that exist in the real world.

4. Choosing a model Type: The right data model type is selected based on the project needs and data properties. Choosing between conceptual, logical, or physical models, or going with a particular model like relational or object‐oriented, may be part of this decision. The degree of abstraction and detail in the representation is determined by the model type that is selected. It guarantees adherence to project objectives and facilitates the development of a model appropriate for the data type.

5. Implementing and Maintaining: The process of implementation converts a physical or logical data model into a database schema. This entails establishing constraints, generating tables, and adding database-specific information. Updating the model to account for shifting technological or commercial needs is called maintenance. Significance: The theoretical model becomes a useful database upon implementation. Frequent upkeep guarantees that the model stays current and accurate, allowing it to adjust to the changing requirements of the company.

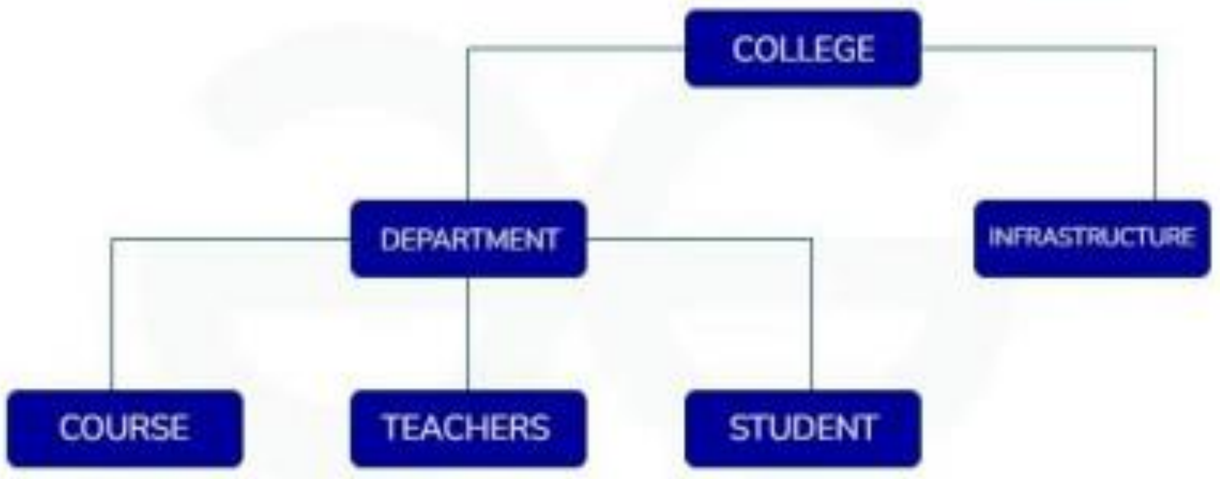
Types of Data Modeling

These are the 5 different types of data models:

Hierarchical Model:

The structure of the hierarchical model resembles a tree. The remaining child nodes are arranged in a certain sequence, and there is only one root node—

or, alternatively, one parent node. However, the hierarchical approach is no longer widely applied. approach connections in the actual world may be modelled using this approach. For Example, for example, in a college there are many courses, many professors and students. So, college became a parent and professors and students became its children.



Relational Model: Relational Mode represent the links between tables by representing data as rows and columns in tables. It is frequently utilised in database design and is strongly related to relational database management systems (RDBMS).

Object-Oriented Data Model: In this model, data is represented as objects, similar to those used in object-oriented programming, creating objects with stored values is the object‑oriented method. In addition to allowing data abstraction, inheritance, and encapsulation, the object-oriented architecture facilitates communication.

Network Model: We have a versatile approach to represent objects and the relationships among these things thanks to the network model. One of its features is a schema, which is a graph representation of the data. An item is stored within a node, and the relationship between them is represented as an edge. This allows them to generalise the maintenance of many parent and child records.

ER-Model: A high-level relational model called the entity-relationship model (ER model) is used to specify the data pieces and relationships between the entities in a system. This conceptual design gives us an easier-to-understand perspective on the facts. An entity‑relationship diagram, which is made up of entities, attributes, and relationships, is used in this model to depict the whole database.

A relationship between entities is called an association. Mapping cardinality many associations like:

• one to one

• one to many

• many to one

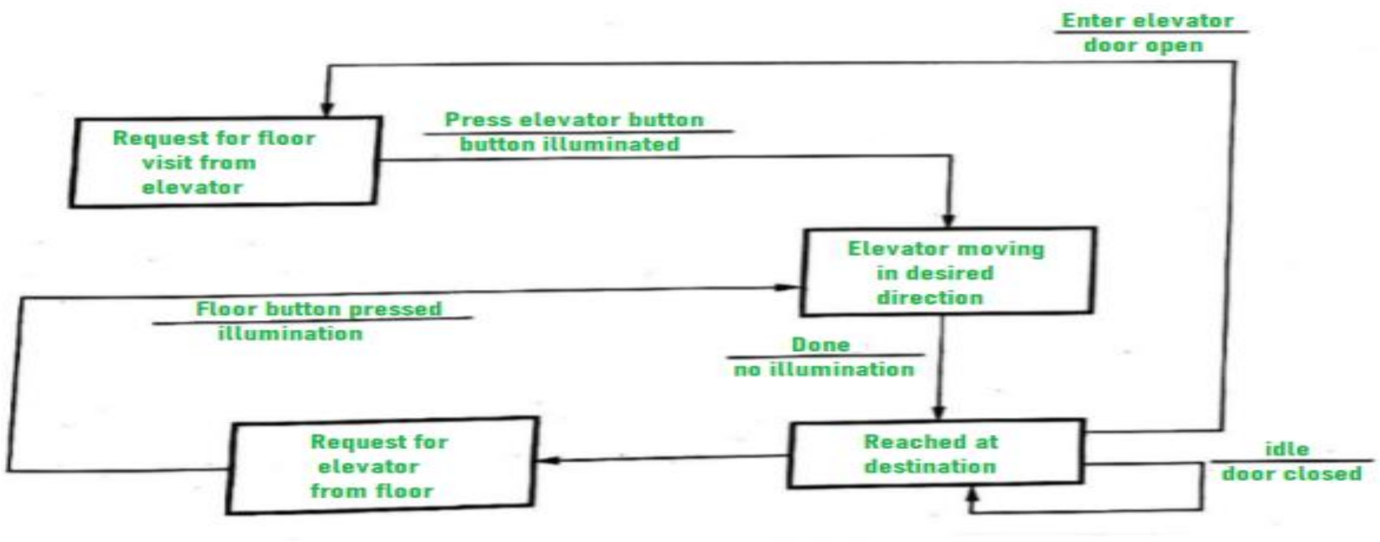
• many to many

Behavioural Model

Overall behaviour of a system can be fully understood by Behavioural model. Behavioural Model is specially designed to make us understand behaviour and factors that influence behaviour of a System. Behaviour of a system is explained and represented with the help of a diagram. This diagram is known as State Transition Diagram. It is a collection of states and events. It usually describes overall states that a system can have and events which are responsible for a change in state of a system. So, on some occurrence of a particular event, an action is taken and what action needs to be taken is represented by State Transition Diagram. Example: Consider an Elevator. This elevator is for n number of floors and has n number of buttons one for each floor. Elevator's working can be explained as follows:

1. Elevator buttons are type of set of buttons which is there on elevator. For reaching a particular floor you want to visit, "elevator buttons" for that particular floor is pressed. Pressing, will cause illumination and elevator will start moving towards that particular floor for which you pressed "elevator buttons". As soon as elevator reaches that particular floor, illumination gets cancelled.

2. Floor buttons are another type of set of buttons on elevator. If a person is on a particular floor and he wants to go on another floor, then elevator button for that floor is pressed. Then, process will be same as given above. Pressing, will cause illumination and elevator to start moving, and when it reaches on desired floor, illumination gets cancelled.
3. When there is no request for elevator, it remains closed on current floor. State Transition Diagram for an elevator system is shown below –



STATE TRANSITION DIAGRAM

Advantages:

• Behaviour and working of a system can easily be understood without any effort.

• Results are more accurate by using this model.

• This model requires less cost for development as cost of resources can be minimal.

• It focuses on behaviour of a system rather than theories.

Disadvantages :

• This model does not have any theory, so trainee is not able to fully understand basic principle and major concept of modelling.

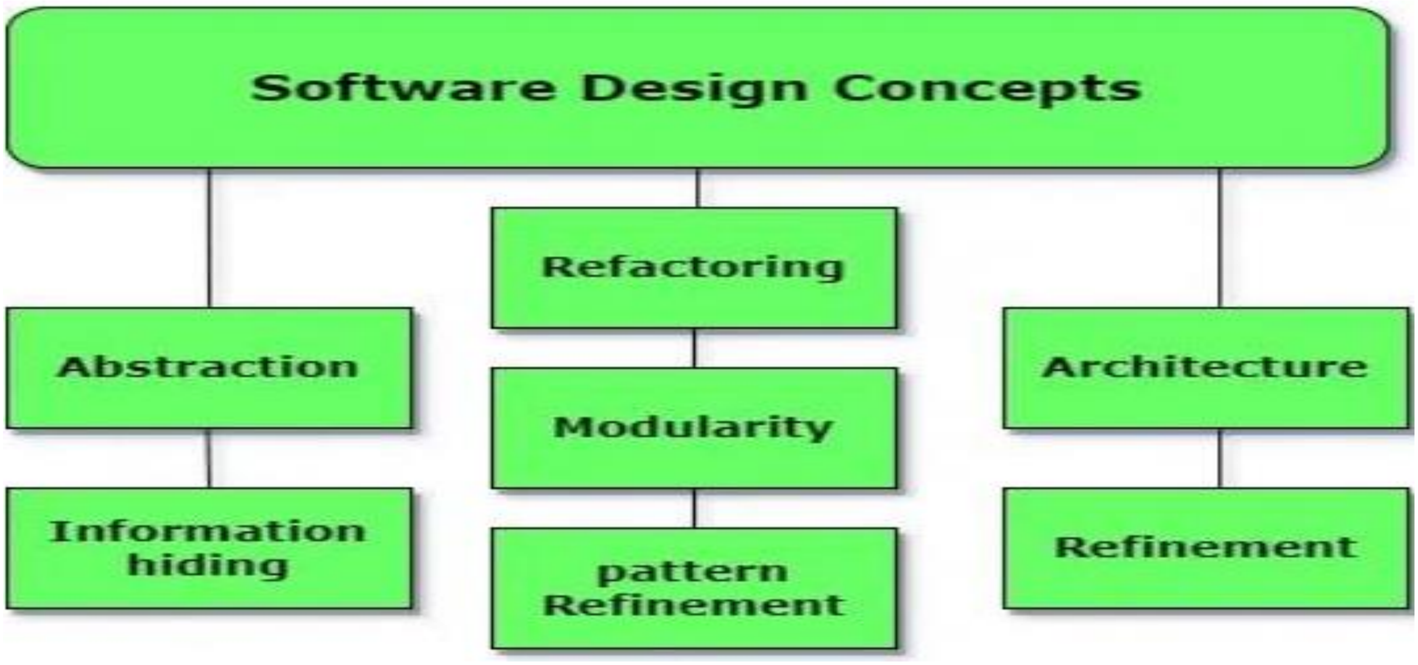
• This modelling cannot be fully automated.

• Sometimes, it's not easy to understand overall result.

• Does not achieve maximum productivity due to some technical issues or any errors.

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, 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. The solution should be described in broad ways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.

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 behaviour of any other features. Refactoring in software design means reconstructing the design to reduce complexity and simplify it without impacting the behaviour or its functions. Fowler has defined refactoring as "the process of changing a software system in a way that it won't impact the behaviour of the design and improves the internal structure".

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:

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

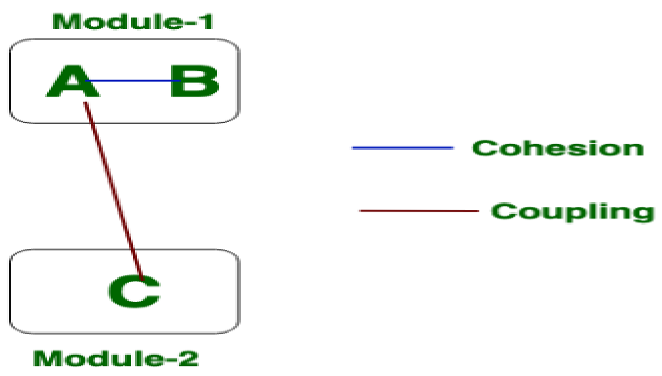
o Coupling: It measures the relative interdependence among modules.

What is Cohesion:

Cohesion is the indication of the relationship within the module. It is the concept of intro-module. Cohesion has many types but usually, high cohesion is good for software.

What is Coupling:

Coupling is also the indication of the relationships between modules. It is the concept of the Inter-module. The coupling has also many types but typically, the low coupling is good for software.



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.
3. 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.

Coupling and Cohesion

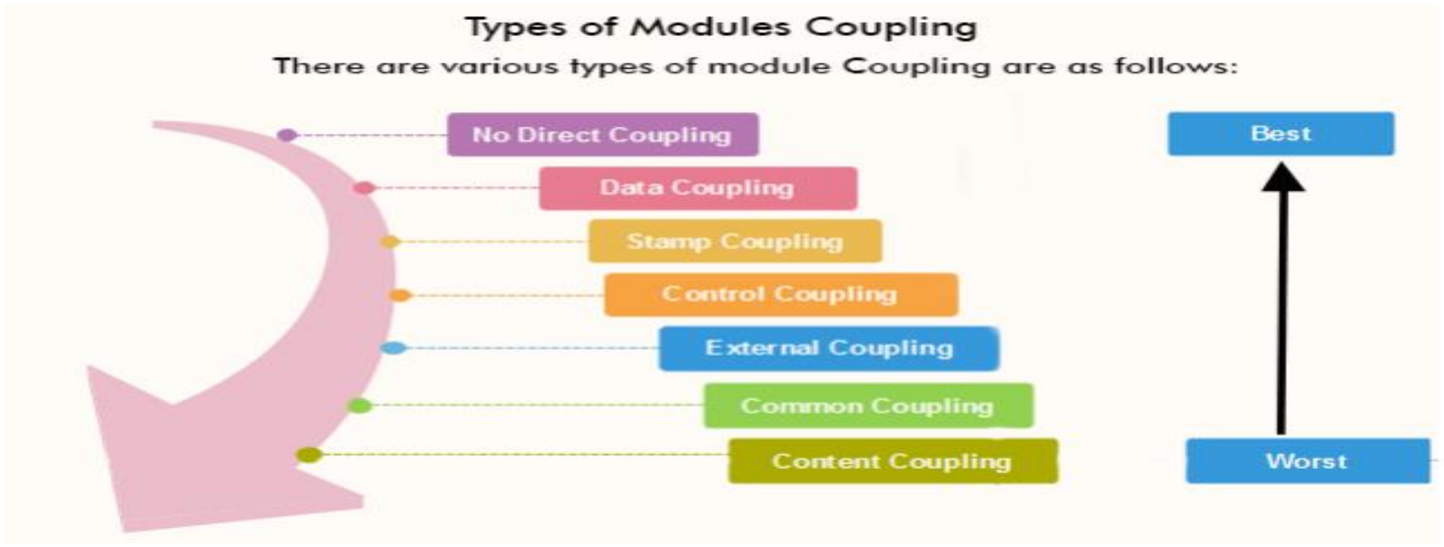
Module Coupling

In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other.

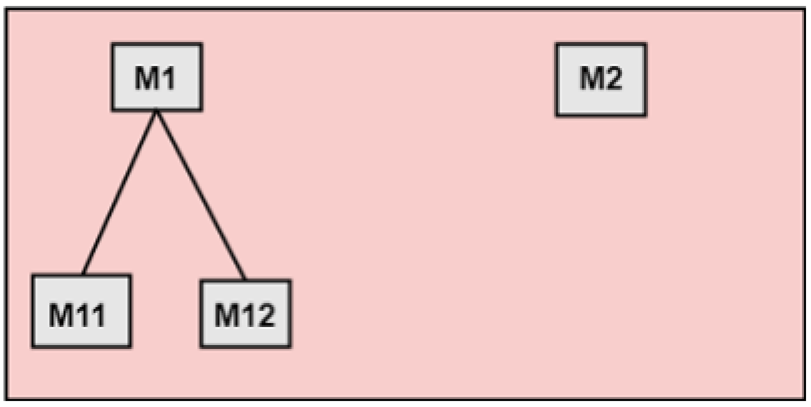
The various types of coupling techniques are shown in fig:

A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have more errors.

Types of Module Coupling

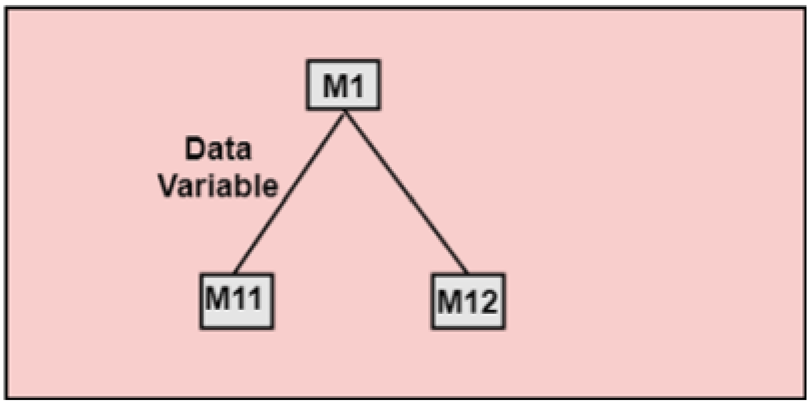


1. No Direct Coupling: There is no direct coupling between M1 and M2.



In this case, modules are subordinates to different modules. Therefore, no directcoupling.

2. Data Coupling: When data of one module is passed to another module, this is called data coupling.

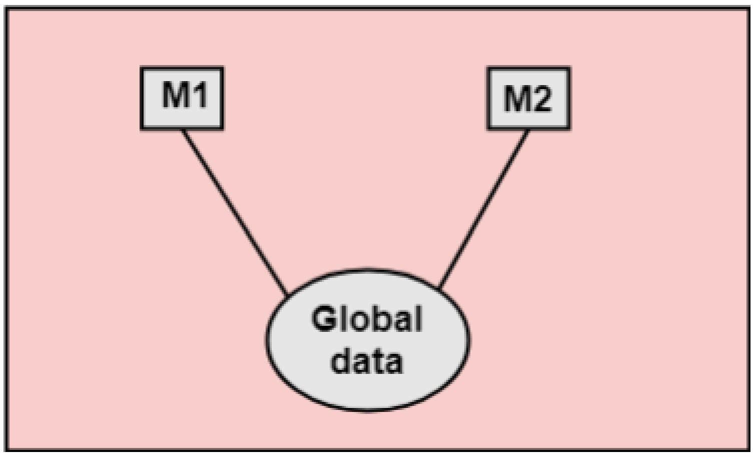


3. Stamp Coupling: Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object in C++ language to a module.

4. Control Coupling: Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.

5. External Coupling: External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.

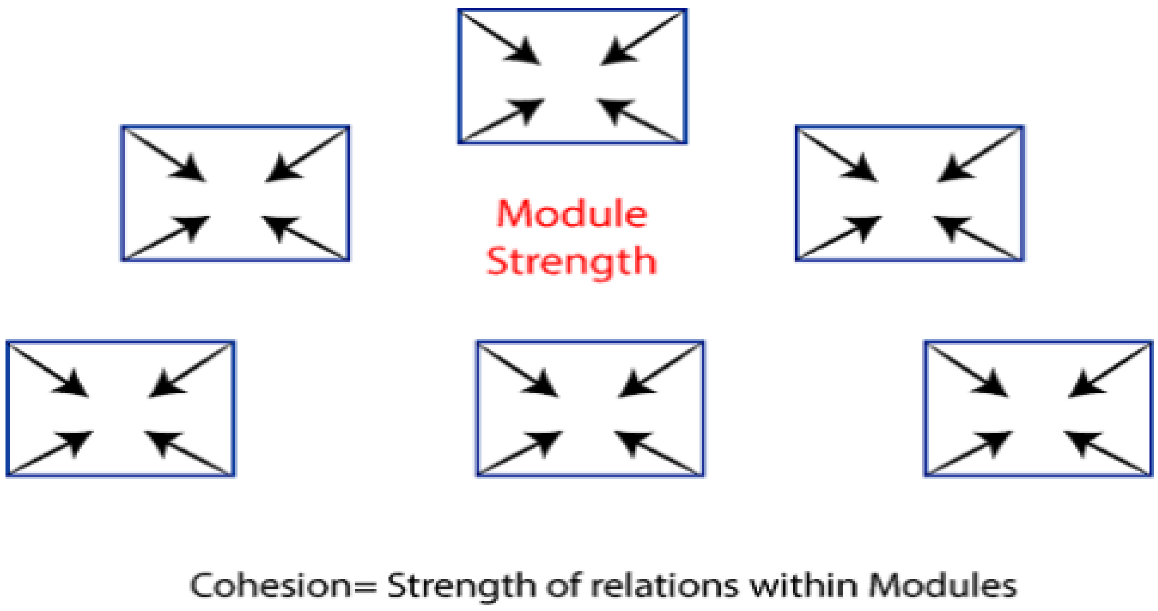
6. Common Coupling: Two modules are common coupled if they share information through some global data items



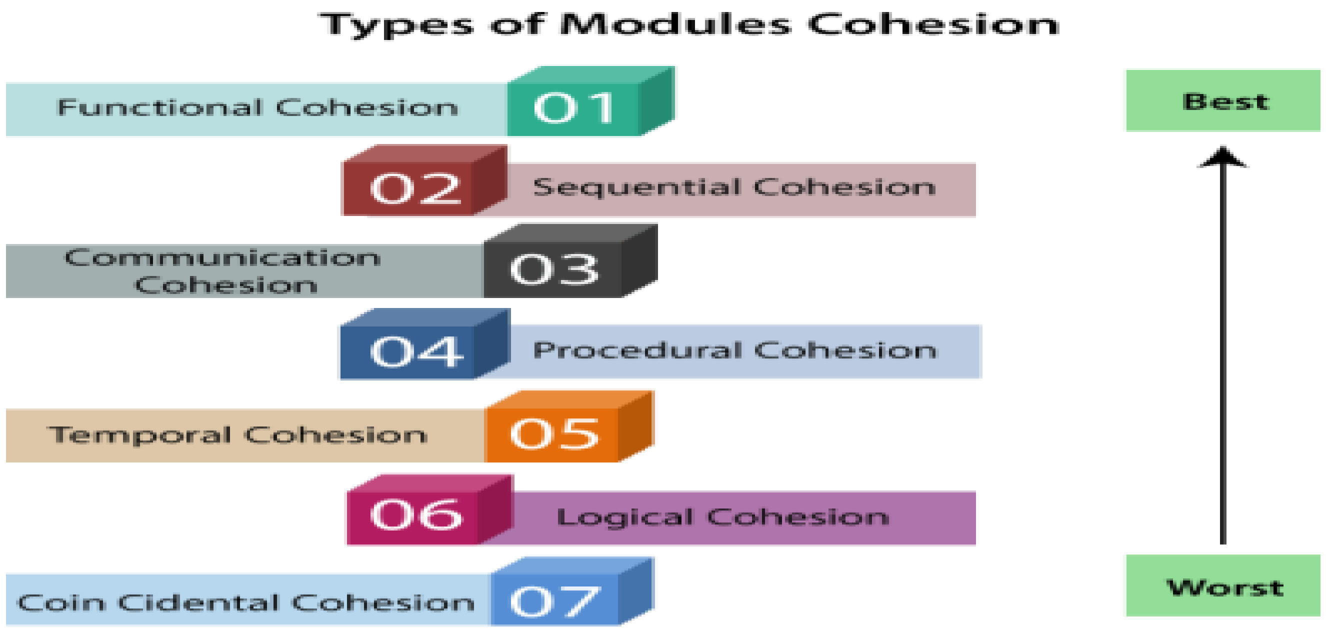
7. Content Coupling: Content Coupling exists among two modules if they sharecode, e.g., a branch from one module into another module.

Module Cohesion

In computer programming, cohesion defines to the degree to which the elements ofa module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related. Cohesion is an ordinal type of measurement and is generally described as "high cohesion" or "low cohesion."



Types of Modules Cohesion



1. Functional Cohesion: Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.

2. Sequential Cohesion: A module is said to possess sequential cohesion if the element of a module form the components of the sequence, where the output from one component of the sequence is input to the next.

3. Communicational Cohesion: A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.

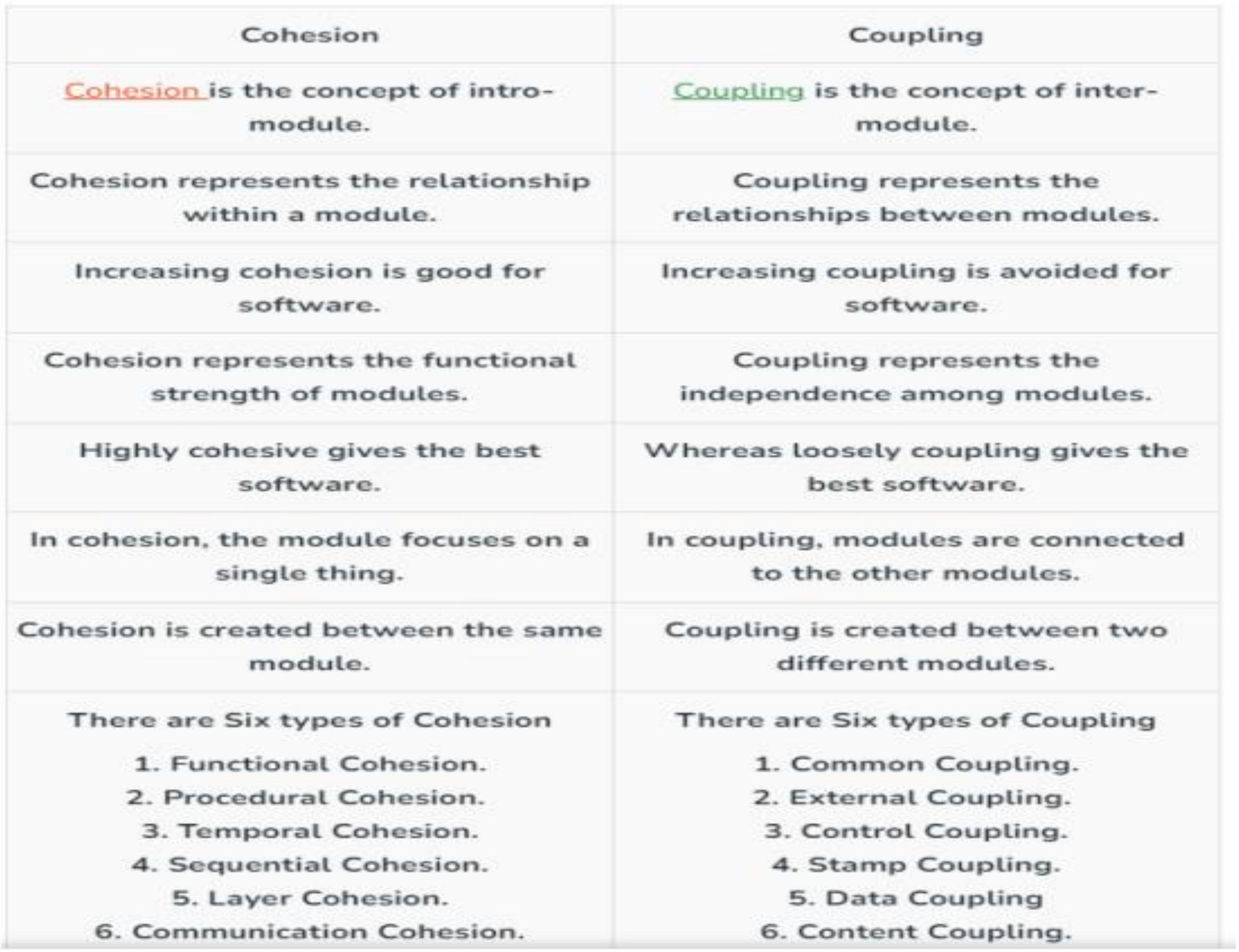
4. Procedural Cohesion: A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.

5. Temporal Cohesion: When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.

6. Logical Cohesion: A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example, Error handling, data input and data output, etc.

7. Coincidental Cohesion: A module is said to have coincidental cohesion if it performs set of tasks that are associated with each other very loosely, if at all

Differentiate between Coupling and Cohesion



Cohesion	Coupling
Cohesion is the concept of intro-module.	Coupling is the concept of inter-module.
Cohesion represents the relationship within a module.	Coupling represents the relationships between modules.
Increasing cohesion is good for software.	Increasing coupling is avoided for software.
Cohesion represents the functional strength of modules.	Coupling represents the independence among modules.
Highly cohesive gives the best software.	Whereas loosely coupling gives the best software.
In cohesion, the module focuses on a single thing.	In coupling, modules are connected to the other modules.
Cohesion is created between the same module.	Coupling is created between two different modules.
There are Six types of Cohesion 1. Functional Cohesion. 2. Procedural Cohesion. 3. Temporal Cohesion. 4. Sequential Cohesion. 5. Layer Cohesion. 6. Communication Cohesion.	There are Six types of Coupling 1. Common Coupling. 2. External Coupling. 3. Control Coupling. 4. Stamp Coupling. 5. Data Coupling 6. Content Coupling.

Top-down and bottom-up design

Top-Down Design Model:

In the top-down model, an overview of the system is formulated without going into detail for any part of it. Each part of it then refined into more details, defining it in yet more details until the entire specification is detailed enough to validate the model. if we glance at a haul as a full, it’s going to appear not possible as a result of it’s so complicated for example: Writing a university system program, writing a word processor. Complicated issues may be resolved victimization high down style, conjointly referred to as Stepwise refinement where,

1. We break the problem into parts,

2. Then break the parts into parts soon and now each of parts will be easy to do.

Advantages:

• Breaking problems into parts help us to identify what needs to be done.

• At each step of refinement, new parts will become less complex and therefore easier to solve.

• Parts of the solution may turn out to be reusable.

• Breaking problems into parts allows more than one person to solve the problem.

Bottom-Up Design Model:

In this design, individual parts of the system are specified in detail. The parts are linked to form larger components, which are in turn linked until a complete system is formed. Object‐oriented language such as C++ or java uses a bottom-up approach where each object is identified first.

Advantage:

• Make decisions about reusable low-level utilities then decide how there will be put together to create high-level construct.,

The contrast between Top-down design and bottom-up design.

S. No.	TOP-DOWN APPROACH	BOTTOM-UP APPROACH
1.	In this approach We focus on breaking up the problem into smaller parts.	In bottom-up approach, we solve smaller problems and integrate it as whole and complete the solution.
2.	Mainly used by structured programming language such as COBOL, Fortran, C, etc.	Mainly used by object-oriented programming language such as C++, C#, Python.
3.	Each part is programmed separately therefore contain redundancy.	Redundancy is minimized by using data encapsulation and data hiding.
4.	In this the communications is less among modules.	In this module must have communication.
5.	It is used in debugging, module documentation, etc.	It is basically used in testing.
6.	In top-down approach, decomposition takes place.	In bottom-up approach composition takes place.
7.	In this top function of system might be hard to identify.	In this sometimes we can not build a program from the piece we have started.
8.	In this implementation details may differ.	This is not natural for people to assemble.
9.	**Pros-** Easier isolation of interface errors • It benefits in the case error occurs towards the top of the program.	**Pros-** • Easy to create test conditions • Test results are easy to observe • It is suited if defects occur at the bottom of the program.

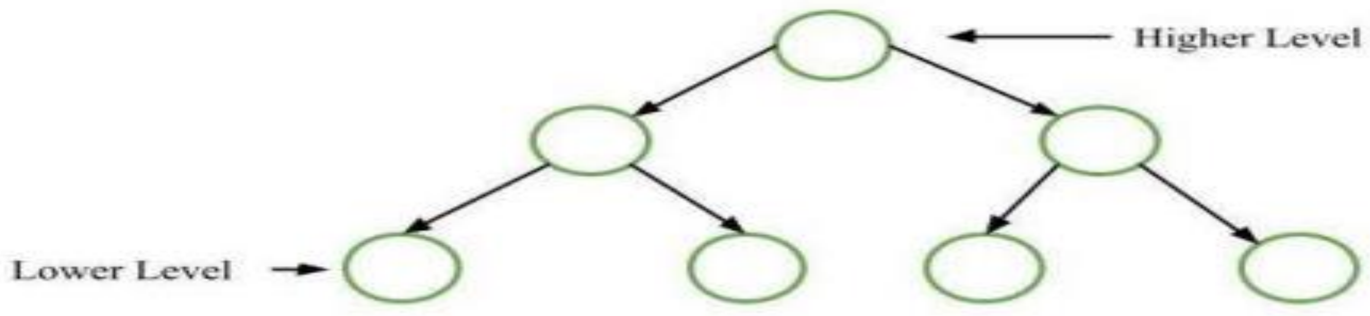
S. No.	TOP-DOWN APPROACH	BOTTOM-UP APPROACH
	• Defects in design get detected early and can be corrected as an early working module of the program is available.	
10.	**Cons-** • Difficulty in observing the output of test case. • Stub writing is quite crucial as it leads to setting of output parameters. • When stubs are located far from the top level module, choosing test cases and designing stubs become more challenging.	**Cons-** • There is no representation of the working model once several modules have been constructed. • There is no existence of the program as an entity without the addition of the last module. • From a partially integrated system, test engineers cannot observe system-level functions. It can be possible only with the installation of the top-level test driver.

Function Oriented Design

The design process for software systems often has two levels. At the first level, the focus is on deciding which modules are needed for the system based on SRS (Software Requirement Specification) and how the modules should be interconnected. Function Oriented Design is an approach to software design where the design is decomposed into a set of interacting units where each unit has a clearly defined function.

Generic Procedure

Start with a high-level description of what the software/program does. Refine each part of the description by specifying in greater detail the functionality of each part. These points lead to a Top-Down Structure.

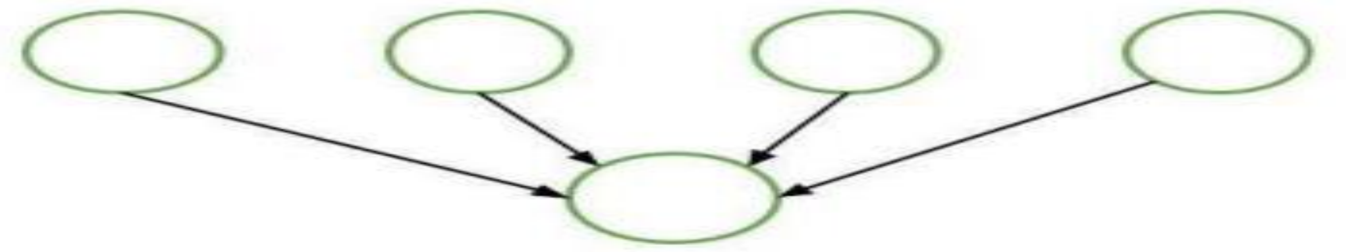


Problem in Top-Down Design Method

Mostly each module is used by at most one other module and that module is called its Parent module.

Solution to the Problem Designing of reusable module.

It means modules use several modules to do their required functions



Function Oriented Design Strategies

Function Oriented Design Strategies are as follows:

1. Data Flow Diagram (DFD): A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

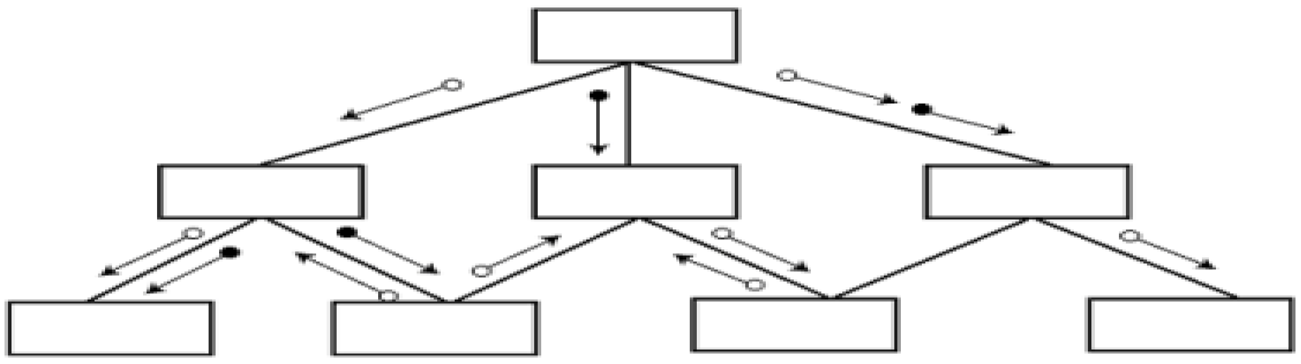
2. Data Dictionaries: Data dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirement stage, data dictionaries contains data items. Data dictionaries include Name of the item, Aliases (Other names for items), Description / purpose, Related data items, Range of values, Data structure definition / form.

3. Structure Charts: Structure chart is the hierarchical representation of system which partitions the system into black boxes (functionality is known to users, but inner details are unknown). Components are read from top to bottom and left to right. When a module calls another, it views the called module as a black box, passing required parameters and receiving results.

4. Pseudo Code: Pseudo Code is system description in short English like phrases describing the function. It uses keyword and indentation. Pseudocodes are used as replacement for flow charts. It decreases the amount of documentation required.

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

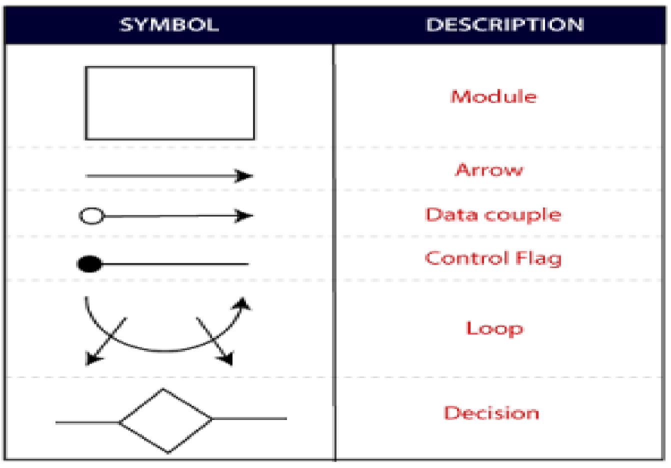
Structured Chart is a graphical representation which shows:

o System partitions into modules o Hierarchy of component modules

o The relation between processing modules o Interaction between modules

o Information passed between modules

The following notations are used in structured chart:



SYMBOL	DESCRIPTION
▭	Module
→	Arrow
○→	Data couple
●—	Control Flag
⟲	Loop
◇	Decision

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 super classes. 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.

Coding

The coding is the process of transforming the design of a system into a computer language format. This coding phase of software development is concerned with software translating design specification into the source code. It is necessary to write source code & internal documentation so that conformance of the code to its specification can be easily verified.

Coding is done by the coder or programmers who are independent people than the designer. The goal is not to reduce the effort and cost of the coding phase, but to cut to the cost of a later stage. The cost of testing and maintenance can be significantly reduced with efficient coding.

Goals of Coding

1. To translate the design of system into a computer language format: The coding is the process of transforming the design of a system into a computer language format, which can be executed by a computer and that perform tasks as specified by the design of operation during the design phase.

2. To reduce the cost of later phases: The cost of testing and maintenance can be significantly reduced with efficient coding.

3. Making the program more readable: Program should be easy to read and understand. It increases code understanding having readability and understandability as a clear objective of the coding activity can itself help in producing more maintainable software.

Characteristics of Programming Language

Following are the characteristics of Programming Language:

Readability: A good high-level language will allow programs to be written in some methods that resemble a quite-English description of the underlying functions. The coding may be done in an essentially self-documenting way.

Portability: High-level languages, being virtually machine-independent, should be easy to develop portable software.

Generality: Most high-level languages allow the writing of a vast collection of programs, thus relieving the programmer of the need to develop into an expert in many diverse languages.

Brevity: Language should have the ability to implement the algorithm with less amount of code. Programs mean in high-level languages are often significantly shorter than their low�level equivalents.

Error checking: A programmer is likely to make many errors in the development of a computer program. Many high-level languages invoke a lot of bugs checking both at compile�time and run-time.

Cost: The ultimate cost of a programming language is a task of many of its characteristics.

Quick translation: It should permit quick translation.

Efficiency: It should authorize the creation of an efficient object code.

Modularity: It is desirable that programs can be developed in the language as several separately compiled modules, with the appropriate structure for ensuring self-consistency among these modules.

Widely available: Language should be widely available, and it should be feasible to provide translators for all the major machines and all the primary operating systems.

A coding standard lists several rules to be followed during coding, such as the way variables are to be named, the way the code is to be laid out, error return conventions, etc.

Coding Standards

General coding standards refers to how the developer writes code, so here we will discuss some essential standards regardless of the programming language being used.

The following are some representative coding standards:

1. Indentation: Proper and consistent indentation is essential in producing easy to read and maintainable programs. Indentation should be used to:

- ➢ Emphasize the body of a control structure such as a loop or a select statement.
- ➢ Emphasize the body of a conditional statement
- ➢ Emphasize a new scope block

2. Inline comments: Inline comments analyse the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.

3. Rules for limiting the use of global: These rules file what types of data can be declared global and what cannot.

4. Structured Programming: Structured (or Modular) Programming methods shall be used. "GOTO" statements shall not be used as they lead to "spaghetti" code, which is hard to read and maintain, except as outlined line in the FORTRAN Standards and Guidelines.

5. Naming conventions for global variables, local variables, and constant identifiers: A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters.

6. Error return conventions and exception handling system: Different functions in a program report the way error conditions are handled should be standard within an organization. For example, different tasks while encountering an error condition should either return a 0 or 1 consistently

Coding Guidelines

General coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain. Most of the examples use the C language syntax, but the guidelines can be tested to all languages.

The following are some representative coding guidelines recommended by many software development organizations.

1. Line Length: It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.

2. Spacing: The appropriate use of spaces within a line of code can improve readability.

Example:

Bad: cost=price+(price*sales_tax)

fprintf(stdout ,"The total cost is %5.2f\n",cost);

Better: cost = price + (price * sales_tax)

fprintf (stdout,"The total cost is %5.2f\n",cost);

3. The code should be well-documented: As a rule of thumb, there must be at least one comment line on the average for every three-source line.

4. The length of any function should not exceed 10 source lines: A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs

5. Do not use goto statements: Use of goto statements makes a program unstructured and very tough to understand.

6. Inline Comments: Inline comments promote readability.

7. Error Messages: Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

Programming Style

Programming style refers to the technique used in writing the source code for a computer program. Most programming styles are designed to help programmers quickly read and understands the program as well as avoid making errors. (Older programming styles also focused on conserving screen space.) A good coding style can overcome the many deficiencies of a first programming language, while poor style can defeat the intent of an excellent language.

The goal of good programming style is to provide understandable, straightforward, elegant code. The programming style used in a various program may be derived from the coding standards or code conventions of a company or other computing organization, as well as the preferences of the actual programmer.

Some general rules or guidelines in respect of programming style:

1. Clarity and simplicity of Expression: The programs should be designed in such a manner so that the objectives of the program is clear.

2. Naming: In a program, you are required to name the module, processes, and variable, and so on. Care should be taken that the naming style should not be cryptic and non-representative.

For Example:

a = 3.14 * r * r

area of circle = 3.14 * radius * radius;

3. Control Constructs: It is desirable that as much as a possible single entry and single exit constructs used.

4. Information hiding: The information secure in the data structures should be hidden from the rest of the system where possible. Information hiding can decrease the coupling between modules and make the system more maintainable.

5. Nesting: Deep nesting of loops and conditions greatly harm the static and dynamic behavior of a program. It also becomes difficult to understand the program logic, so it is desirable to avoid deep nesting.

6. User-defined types: Make heavy use of user-defined data types like enum, class, structure, and union. These data types make your program code easy to write and easy to understand.

7. Module size: The module size should be uniform. The size of the module should not be too big or too small. If the module size is too large, it is not generally functionally cohesive. If the module size is too small, it leads to unnecessary overheads.

8. Module Interface: A module with a complex interface should be carefully examined.

9. Side-effects: When a module is invoked, it sometimes has a side effect of modifying the program state. Such side-effect should be avoided where as possible.