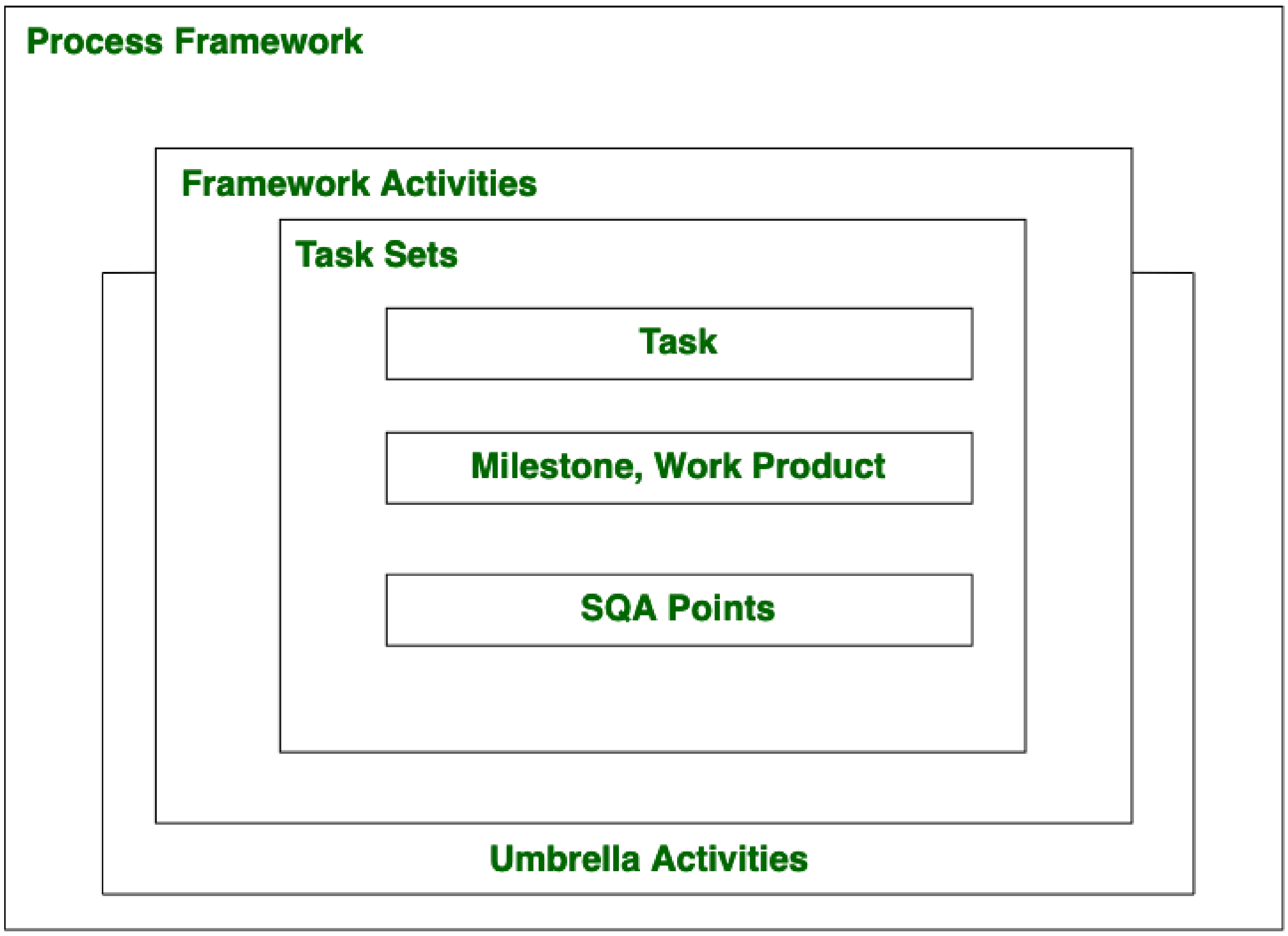
Software Design is the process to transform the user requirements into some suitable form, which helps the programmer in software coding and implementation.
During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence the aim of this phase is to transform the SRS document into the design document.

5.1 Objective of Design

Correctness:
A good design should be correct i.e. it should correctly implement all the functionalities of the system.
Efficiency:
A good software design should address the resources, time, and cost optimization issues.
Understandability:
A good design should be easily understandable, for which it should be modular and all the modules are arranged in layers.
Completeness:
The design should have all the components like data structures, modules, and external interfaces, etc.
Maintainability:
A good software design should be easily amenable to change whenever a change request is made from the customer side.

5.2 Design Framework

Framework is a Standard way to build and deploy applications. Software Process Framework is the foundation of complete software engineering process. Software process framework includes set of all umbrella activities. It also includes number of framework activities that are applicable to all software projects.



Software Process Framework

A generic process framework encompasses five activities which are given below one by one:

Communication: In this activity, heavy communication with customers and other stakeholders, as well as requirement gathering is done.
Planning: In this activity, we discuss the technical related tasks, work schedule, risks, required resources, etc.
Modeling: Modeling is about building representations of things in the ‘real world’. In modeling activity, a product’s model is created in order to better understand the requirements.

Construction: In software engineering, construction is the application of set of procedures that are needed to assemble the product. In this activity, we generate the code and test the product in order to make better product.
Deployment: In this activity, a complete or non-complete product or software is represented to the customers to evaluate and give feedback. On the basis of their feedback, we modify the product for supply of better product.

Umbrella activities include:

- Risk Management
- Software Quality Assurance (SQA)
- Software Configuration Management (SCM)
- Measurement
- Formal Technical Reviews (FTR)

5.3 Software design models

Design modeling in software engineering represents the features of the software that helps engineer to develop it effectively, the architecture, the user interface, and the component level detail. Design modeling provides a variety of different views of the system like architecture plan for home or building. Different methods like data-driven, pattern-driven, or object-oriented methods are used for constructing the design model. All these methods use set of design principles for designing a model.

Working of Design Modeling in Software Engineering

Designing a model is an important phase and is a multi-process that represent the data structure, program structure, interface characteristic, and procedural details. It is mainly classified into four categories

Data design: It represents the data objects and their interrelationship in an entity-relationship diagram. Entity-relationship consists of information required for each entity or data objects as well as it shows the relationship between these objects. It shows the structure of the data in terms of the tables. It shows three type of relationship – One to one, one to many, and many to many.

Architectural design: It defines the relationship between major structural elements (integration, and testing of the software product providing successive levels of functionality) of the software. It is about decomposing the system into interacting components. It is expressed as a block diagram defining an overview of the system structure – features of the components and how these components communicate with each other to share data. It defines the structure and properties of the component that are involved in the system and also the inter-relationship among these components.

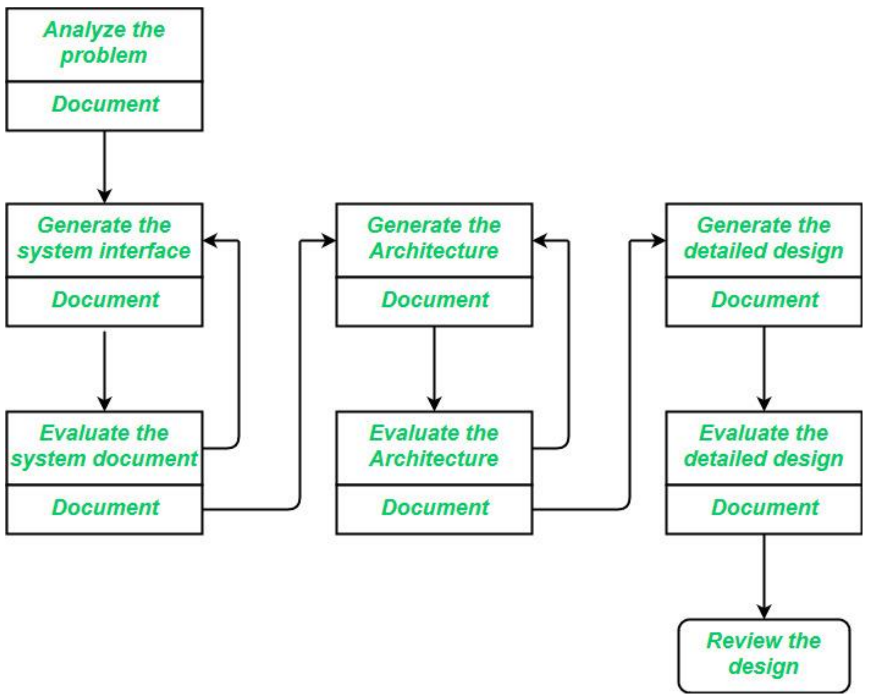
User Interfaces design: It represents how the Software communicates with the user i.e. the behaviour of the system. It refers to the product where users interact with controls or displays of the product. For example, Military, vehicles, aircraft, audio equipment, computer peripherals are the areas where user interface design is implemented

Component level design: It transforms the structural elements of the software architecture into a procedural description of software components. It is a perfect way to share a large amount of data. Components need not be concerned with how data is managed at a centralized level i.e. components need not worry about issues like backup and security of the data.

5.4 Design process

The software design process can be divided into the following three levels of phases of design:

- Interface Design
- Architectural Design
- Detailed Design


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.

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.

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

5.5 Architecture design

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".

An architectural design performs the following functions.

- It evaluates all top-level designs.

- It develops and documents top-level design for the external and internal interfaces.
- It develops preliminary versions of user documentation.
- It defines and documents preliminary test requirements and the schedule for software integration.

Architectural design is of crucial importance in software engineering during which the essential requirements like reliability, cost, and performance are dealt with.

All stakeholders must also be consulted while reviewing the architectural design in order to minimize the risks and errors.

Architectural design can be represented using the following models.

- Structural model: Illustrates architecture as an ordered collection of program components
- Dynamic model: Specifies the behavioral aspect of the software architecture and indicates how the structure or system configuration changes as the function changes due to change in the external environment
- Process model: Focuses on the design of the business or technical process, which must be implemented in the system
- Functional model: Represents the functional hierarchy of a system

Architectural Design Output

The architectural design process results in an Architectural Design Document (ADD). This document consists of a number of graphical representations that comprises software models along with associated descriptive text.

Framework model: Attempts to identify repeatable architectural design patterns encountered in similar types of application. This leads to an increase in the level of abstraction.

Architectural Styles

Architectural styles define a group of interlinked systems that share structural and semantic properties. In short, the objective of using architectural styles is to establish a structure for all the components present in a system.

A computer-based system (software is part of this system) exhibits one of the many available architectural styles. Every architectural style describes a system category that includes the following.

- **Computational components** such as clients, server, filter, and database to execute the desired system function
- A set of **connectors** such as procedure call, events broadcast, database protocols, and pipes to provide communication among the computational components
- **Constraints** to define integration of components to form a system
- A **semantic model**, which enable the software designer to identify the characteristics of the system as a whole by studying the characteristics of its components.

5.6 Low Level Design

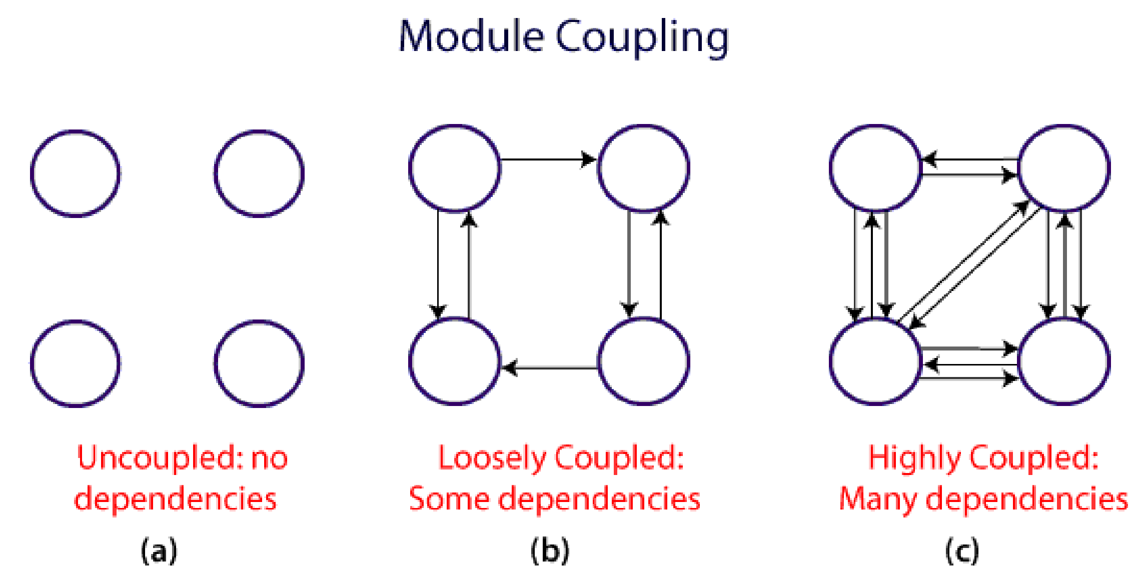
- The LLD stands for Low-Level Design, in which the designer will focus on the components like a User interface (UI).
- The Low-level design is created by the developer manager and designers.
- It is also known as micro-level or detailed design. The LLD can change the High-Level Solution into a detailed solution.
- The Low-level design specifics the detailed description of all modules, which implies that the LLD involves all the system component's actual logic. It goes deep into each module's specification.
- The Low-level design is created after the High-Level Design.

5.7 Coupling and cohesion

Module Coupling

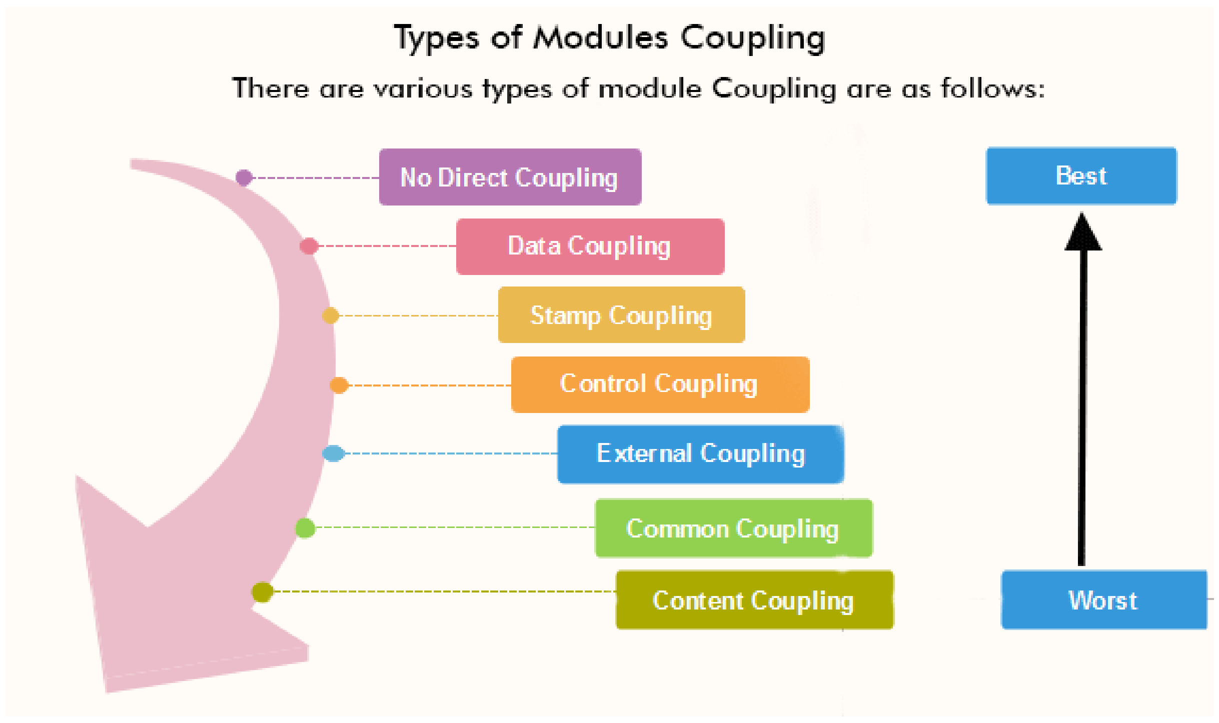
In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. **Uncoupled modules** have no interdependence at all within them.

The various types of coupling techniques are shown in fig:

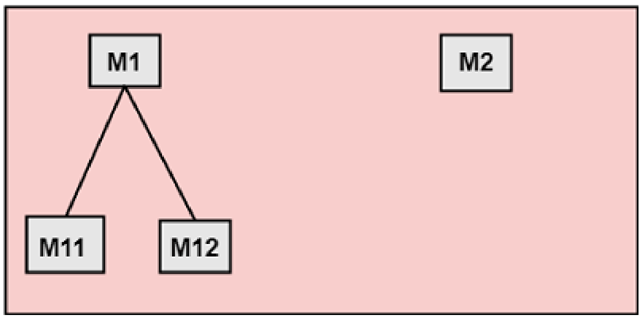

Module Coupling
Uncoupled: no dependencies
(a)
Loosely Coupled: Some dependencies
(b)
Highly Coupled: Many dependencies
(c)

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

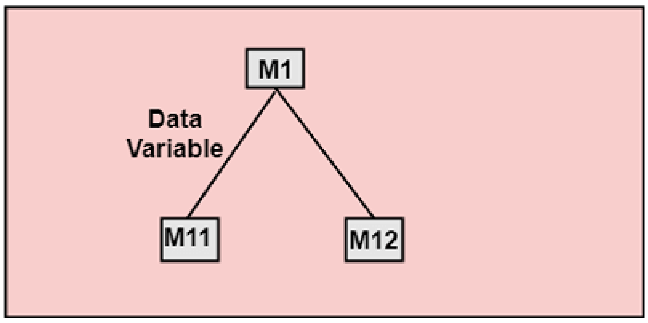


1. No Direct Coupling: There is no direct coupling between M1 and M2.



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

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

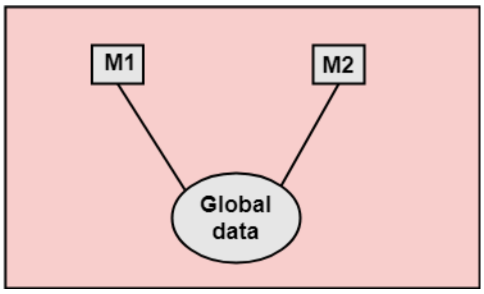


3. Stamp Coupling: Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object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

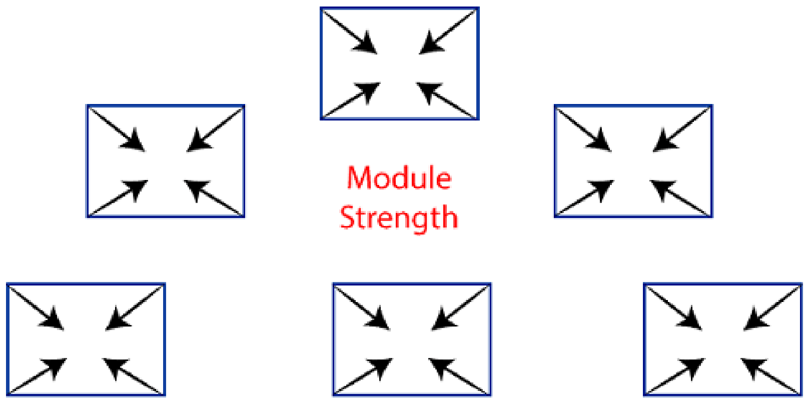


7. Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

Module Cohesion

In computer programming, cohesion defines to the degree to which the elements of a 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."



Cohesion= Strength of relations within Modules

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 a set of tasks that are associated with each other very loosely, if at all.

Differentiate between Coupling and Cohesion

Coupling	**Cohesion**
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.
Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.
Coupling shows the relative **independence** between the modules.	Cohesion shows the module's relative **functional** strength.
While creating, you should aim for low coupling, i.e., dependency among modules should be less.	While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction

	with other modules of the system.
In coupling, modules are linked to the other modules.	In cohesion, the module focuses on a single thing.

5.8 Software design strategies

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

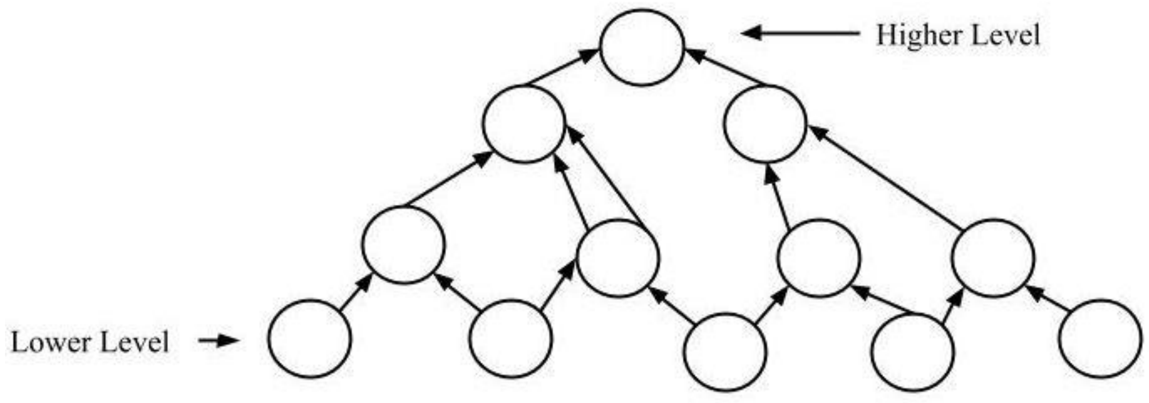
1. If any pre-existing code needs to be understood, organized, and pieced together.
2. It is common for the project team to have to write some code and produce original programs that support the application logic of the system.

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:

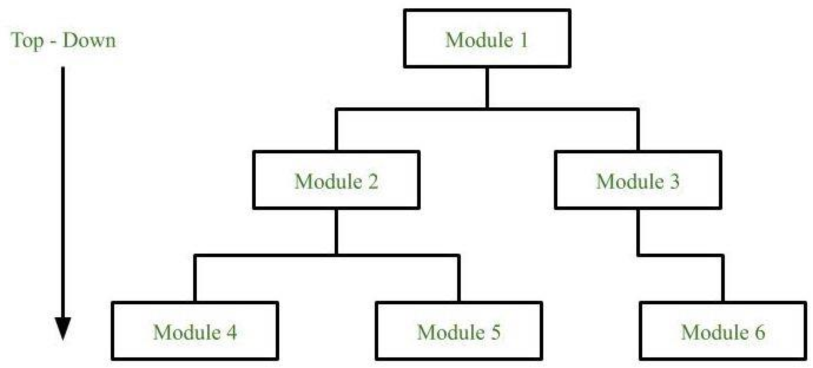
- It is not so closely related to the structure of the problem.
- High-quality bottom-up solutions are very hard to construct.
- It leads to the proliferation of 'potentially useful' functions rather than the most appropriate ones.

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.

5.9 Function 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 functions changes data and state of entire system.
- 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.

5.10 Object oriented design

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.

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.

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.

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.

5.11 Function oriented design Vs Object oriented design

COMPARISON FACTORS	FUNCTION ORIENTED DESIGN	OBJECT ORIENTED DESIGN
Abstraction	The basic abstractions, which are given to the user, are real world functions.	The basic abstractions are not the real world functions but are the data abstraction where the real world entities are represented.
Function	Functions are grouped together by which a higher level function is obtained.	Function are grouped together on the basis of the data they operate since the classes are associated with their methods.
State information	In this approach the state information is often represented in a centralized shared memory.	In this approach the state information is not represented is not represented in a centralized memory but is implemented or distributed among the objects of the system.
Approach	It is a top down approach.	It is a bottom up approach.
Begins basis	Begins by considering the use case diagrams and the scenarios.	Begins by identifying objects and classes.
Decompose	In function oriented design we decompose in function/procedure level.	We decompose in class level.
Use	This approach is mainly used for computation sensitive application.	This approach is mainly used for evolving system which mimics a business or business case.