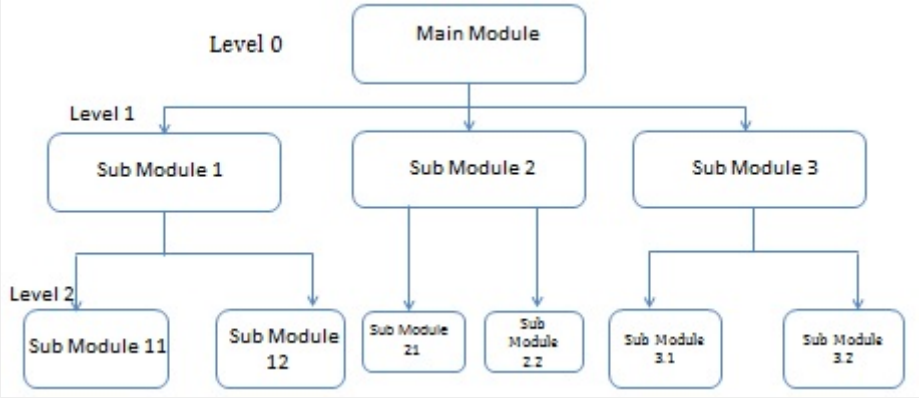
# Design Strategies

## Top-Down Strategy

The top-down strategy uses the modular approach to develop the design of a system. It is called so because it starts from the top or the highest-level module and moves towards the lowest level modules.

In this technique, the highest-level module or main module for developing the software is identified. The main module is divided into several smaller and simpler submodules or segments based on the task performed by each module. Then, each submodule is further subdivided into several submodules of next lower level. This process of dividing each module into several submodules continues until the lowest level modules, which cannot be further subdivided, are not identified.



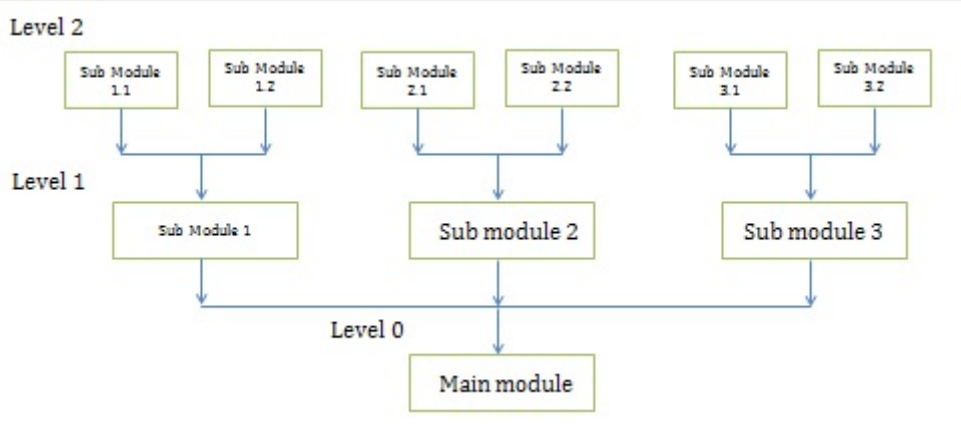

## Bottom-Up Strategy

Bottom-Up Strategy follows the modular approach to develop the design of the system. It is called so because it starts from the bottom or the most basic level modules and moves towards the highest level modules.

In this technique,

- The modules at the most basic or the lowest level are identified.
- These modules are then grouped together based on the function performed by each module to form the next higher-level modules.

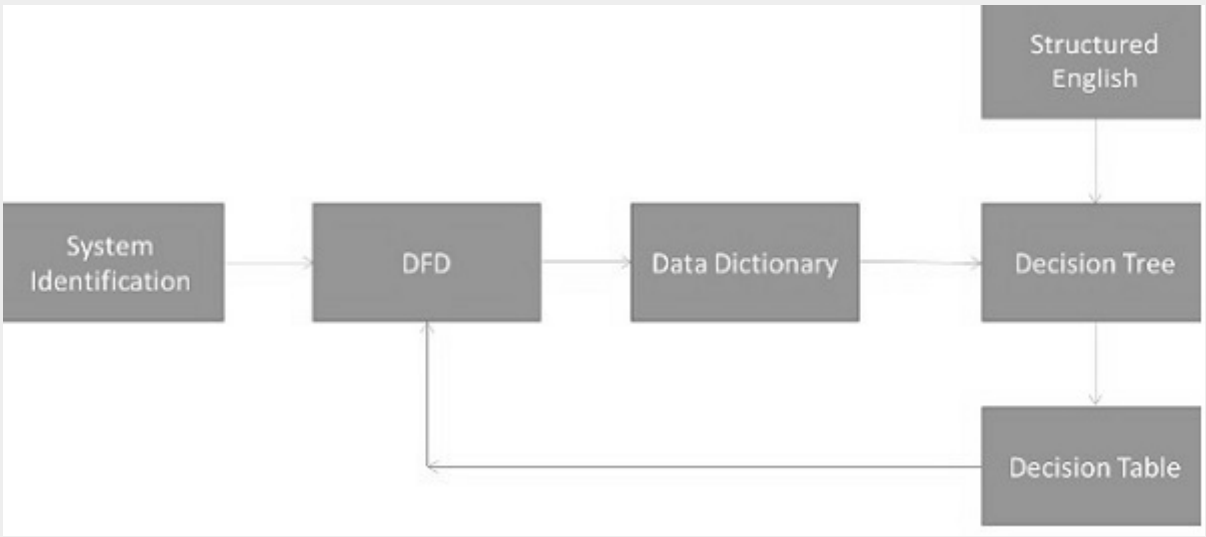
- Then, these modules are further combined to form the next higher-level modules.
- This process of grouping several simpler modules to form higher level modules continues until the main module of system development process is achieved.




# Structured Design

Structured design is a data-flow based methodology that helps in identifying the input and output of the developing system. The main objective of structured design is to minimize the complexity and increase the modularity of a program. Structured design also helps in describing the functional aspects of the system.

In structured designing, the system specifications act as a basis for graphically representing the flow of data and sequence of processes involved in a software development with the help of DFDs. After developing the DFDs for the software system, the next step is to develop the structure chart.




## Modularization

Structured design partitions the program into small and independent modules. These are organized in top down manner with the details shown in bottom.

Thus, structured design uses an approach called Modularization or decomposition to minimize the complexity and to manage the problem by subdividing it into smaller segments.

### Advantages

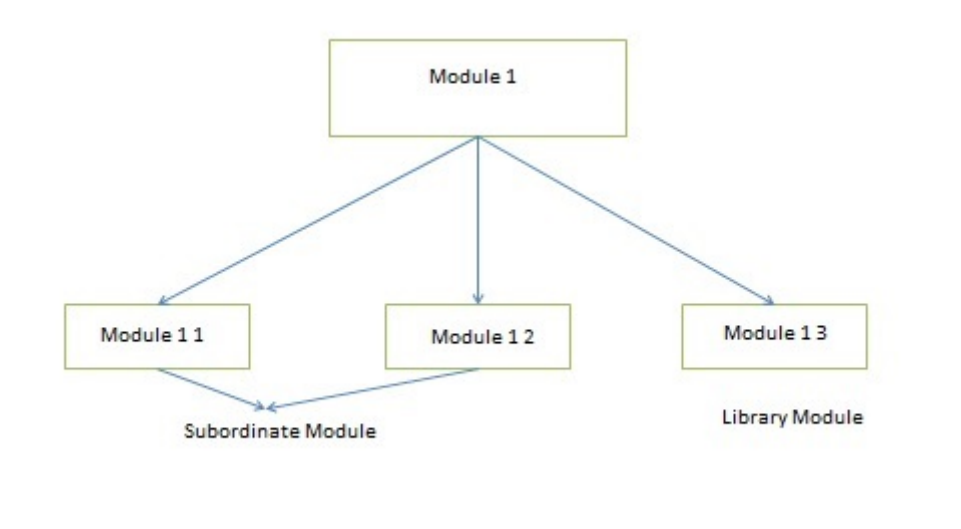
- Critical interfaces are tested first.
- It provide abstraction.
- It allows multiple programmers to work simultaneously.
- It allows code reuse.
- It provides control and improves morale.
- It makes identifying structure easier.

## Structured Charts

Structured charts are a recommended tool for designing a modular, top down systems which define the various modules of system development and the relationship between each module. It shows the system module and their relationship between them.

It consists of diagram consisting of rectangular boxes that represent the modules, connecting arrows, or lines.

- **Control Module** – It is a higher-level module that directs lower-level modules, called **subordinate modules**.
- **Library Module** – It is a reusable module and can be invoked from more than one point in the chart.




We have two different approaches to design a structured chart –

- **Transform-Centered Structured Charts** – They are used when all the transactions follow same path.
- **Transaction–Centered Structured Charts** – They are used when all the transactions do not follow the same path.

## Objectives of Using Structure Flowcharts

- To encourage a top-down design.

- To support the concept of modules and identify the appropriate modules.
- To show the size and complexity of the system.
- To identify the number of readily identifiable functions and modules within each function.
- To depict whether each identifiable function is a manageable entity or should be broken down into smaller components.

# Factors Affecting System Complexity

To develop good quality of system software, it is necessary to develop a good design. Therefore, the main focus on while developing the design of the system is the quality of the software design. A good quality software design is the one, which minimizes the complexity and cost expenditure in software development.

The two important concepts related to the system development that help in determining the complexity of a system are **coupling** and **cohesion**.

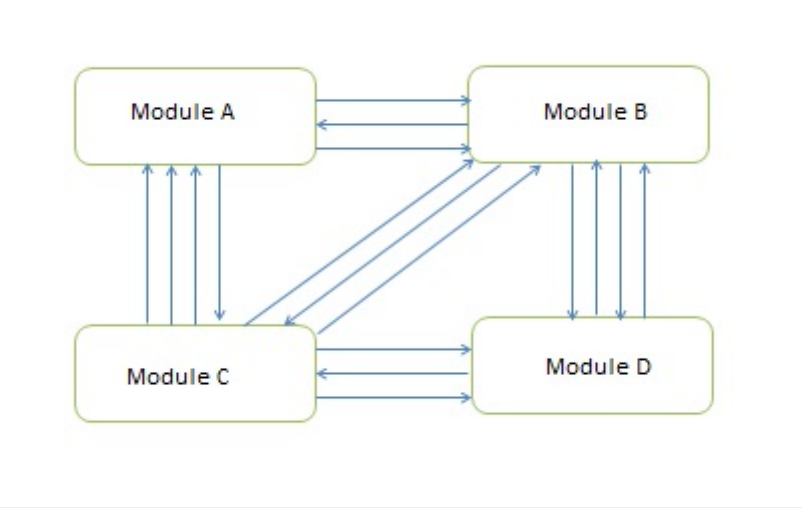
## Coupling

Coupling is the measure of the independence of components. It defines the degree of dependency of each module of system development on the other. In practice, this means the stronger the coupling between the modules in a system, the more difficult it is to implement and maintain the system.

Each module should have simple, clean interface with other modules, and that the minimum number of data elements should be shared between modules.

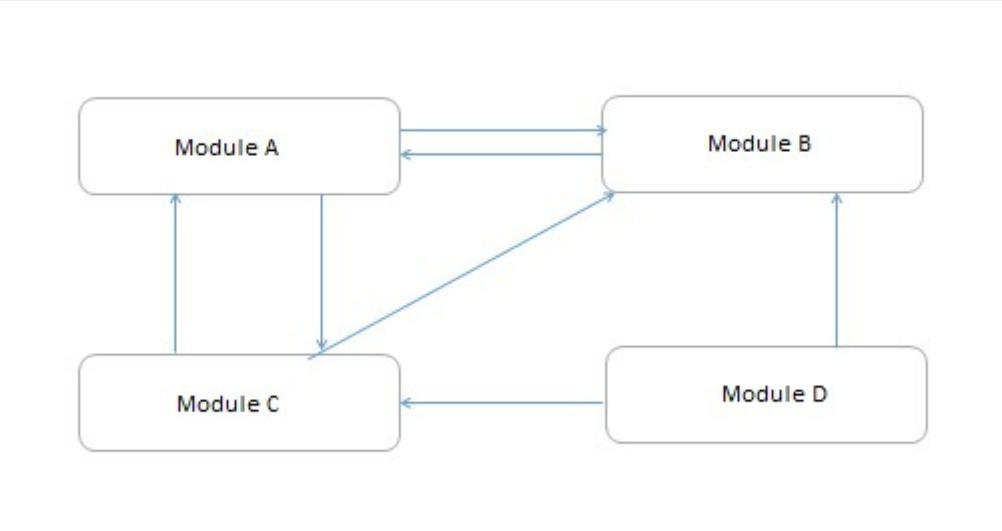
## High Coupling

These type of systems have interconnections with program units dependent on each other. Changes to one subsystem leads to high impact on the other subsystem.



## Low Coupling

These type of systems are made up of components which are independent or almost independent. A change in one subsystem does not affect any other subsystem.




## Coupling Measures

- **Content Coupling** – When one component actually modifies another,then the modified component is completely dependent on modifying one.
- **Common Coupling** – When amount of coupling is reduced somewhat by organizing system design so that data are accessible from a common data store.
- **Control Coupling** – When one component passes parameters to control the activity of another component.
- **Stamp Coupling** – When data structures is used to pass information from one component to another.
- **Data Coupling** – When only data is passed then components are connected by this coupling.

## Cohesion

Cohesion is the measure of closeness of the relationship between its components. It defines the amount of dependency of the components of a module on one another. In practice, this means the systems designer must ensure that –

- They do not split essential processes into fragmented modules.
- They do not gather together unrelated processes represented as processes on the DFD into meaningless modules.

The best modules are those that are functionally cohesive. The worst modules are those that are coincidentally cohesive.

## The worst degree of cohesion

Coincidental cohesion is found in a component whose parts are unrelated to another.

- **Logical Cohesion** – It is where several logically related functions or data elements are placed in same component.
- **Temporal Cohesion** – It is when a component that is used to initialize a system or set variables performs several functions in sequence, but the functions are related by timing involved.
- **Procedurally Cohesion** – It is when functions are grouped together in a component just to ensure this order.
- **Sequential Cohesion** – It is when the output from one part of a component is the input to the next part of it.