# Unit-4

# System Design

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.

Software Engineering is the process of designing, building, testing, and maintaining software. The goal of software engineering is to create software that is reliable, efficient, and easy to maintain. System design is a critical component of software engineering and involves making decisions about the architecture, components, modules, interfaces, and data for a software system.

**System Design Strategy refers to the approach that is taken to design a software system. There are several strategies that can be used to design software systems, including the following:**

1. Top-Down Design: This strategy starts with a high-level view of the system and gradually breaks it down into smaller, more manageable components.
2. Bottom-Up Design: This strategy starts with individual components and builds the system up, piece by piece.
3. Iterative Design: This strategy involves designing and implementing the system in stages, with each stage building on the results of the previous stage.
4. Incremental Design: This strategy involves designing and implementing a small part of the system at a time, adding more functionality with each iteration.
5. Agile Design: This strategy involves a flexible, iterative approach to design, where requirements and design evolve through collaboration between self-organizing and cross-functional teams.

The choice of system design strategy will depend on the particular requirements of the software system, the size and complexity of the system, and the development methodology being used. A well-designed system can simplify the development process, improve the quality of the software, and make the software easier to maintain.

**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.

# https://media.geeksforgeeks.org/wp-content/uploads/20190326154349/Untitled-document10.jpeg

**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 forming a system hierarchy structure. The complete software system is considered a single entity and in relation to the characteristics, the system is split into sub-systems and components. 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, a top-down design best suits the purpose.

# https://media.geeksforgeeks.org/wp-content/uploads/20190326161031/Untitled-document12.jpeg

**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 the 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 top-down and bottom-up design strategies. In this, we can reuse the modules.

**Advantages of using a System Design Strategy:**

1. Improved quality: A well-designed system can improve the overall quality of the software, as it provides a clear and organized structure for the software.
2. Ease of maintenance: A well-designed system can make it easier to maintain and update the software, as the design provides a clear and organized structure for the software.
3. Improved efficiency: A well-designed system can make the software more efficient, as it provides a clear and organized structure for the software that reduces the complexity of the code.
4. Better communication: A well-designed system can improve communication between stakeholders, as it provides a clear and organized structure for the software that makes it easier for stakeholders to understand and agree on the design of the software.
5. Faster development: A well-designed system can speed up the development process, as it provides a clear and organized structure for the software that makes it easier for developers to understand the requirements and implement the software.

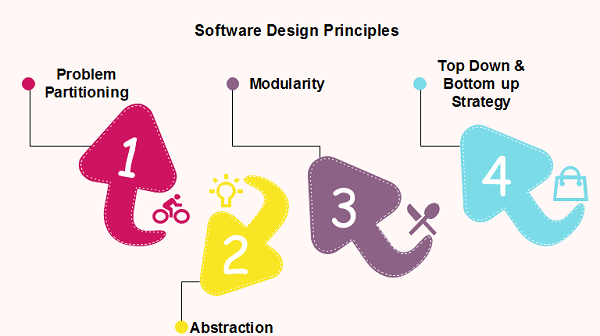
### Disadvantages of using a System Design Strategy:

1. Time-consuming: Designing a system can be time-consuming, especially for large and complex systems, as it requires a significant amount of documentation and analysis.
2. Inflexibility: Once a system has been designed, it can be difficult to make changes to the design, as the process is often highly structured and documentation-intensive

# 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.

## 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

1. A module is specified by the method it performs.
2. 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.

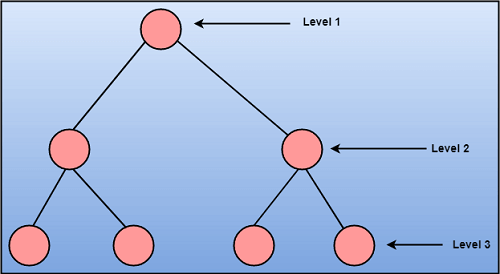
## Strategy of Design

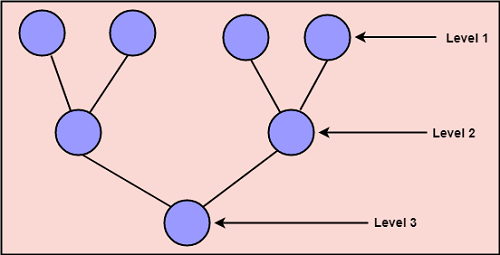
A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed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



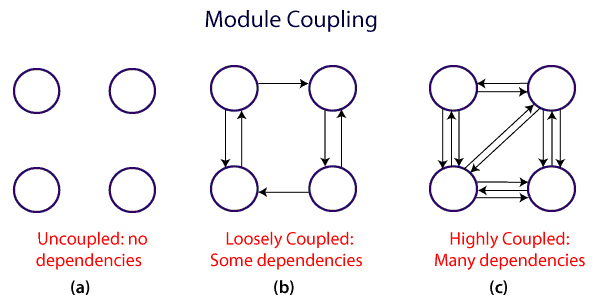
1. **Bottom-up Approach:** A bottom-up approach begins with the lower details and moves towards up the hierarchy, as shown in fig. This approach is suitable in case of an existing system.
2. 

# Coupling and Cohesion

## Module Coupling

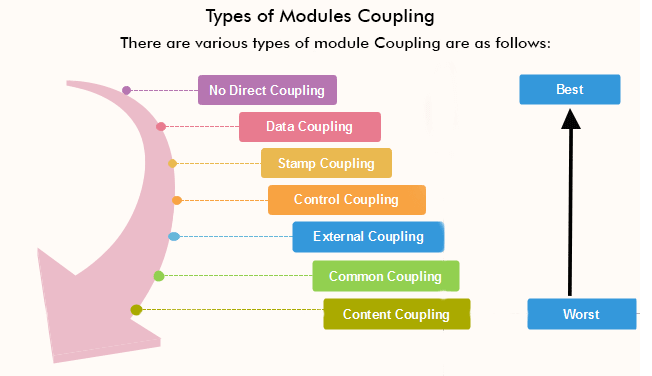
In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. **Uncoupled modules** have no interdependence at all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

**The v 1 arious types of coupling techniques are shown in fig:**

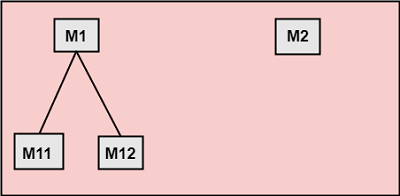


A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have more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

### Types of Module Coupling

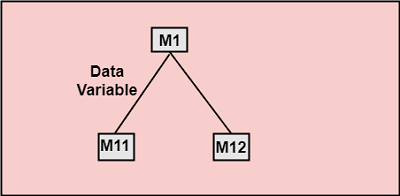


* 1. **No Direct Coupling:** There is no direct coupling between M1 and M2.



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

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

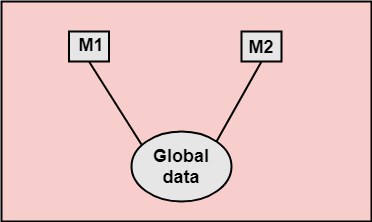


**Stamp Coupling:** Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or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

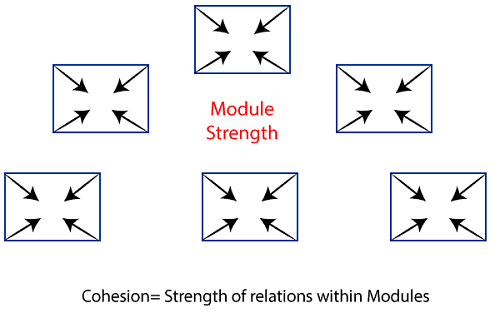


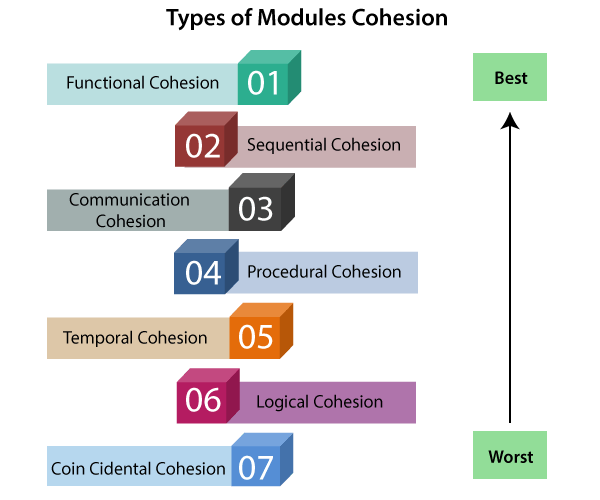
1. **Content Coupling:** Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

## 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."





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. |

# Software Architecture

# The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.

# Software Architecture Types

We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

## Software Architecture

Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

* It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
* Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of −
  + Selection of structural elements and their interfaces by which the system is composed.
  + Behavior as specified in collaborations among those elements.
  + Composition of these structural and behavioral elements into large subsystem.
  + Architectural decisions align with business objectives.
  + Architectural styles guide the organization.

## Software Design

Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows −

* To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.
* Act as a blueprint during the development process.
* Guide the implementation tasks, including detailed design, coding, integration, and testing.

It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.

# Software Design

## Goals of Architecture

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements.

Some of the other goals are as follows −

* Expose the structure of the system, but hide its implementation details.
* Realize all the use-cases and scenarios.
* Try to address the requirements of various stakeholders.
* Handle both functional and quality requirements.
* Reduce the goal of ownership and improve the organization’s market position.
* Improve quality and functionality offered by the system.
* Improve external confidence in either the organization or system.

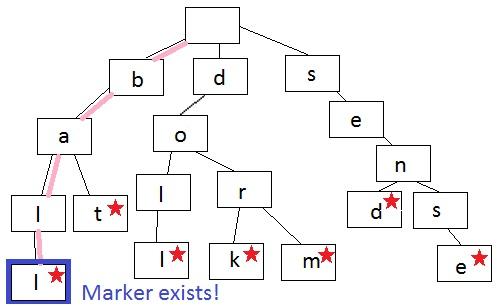
#### CONTROL HIERARCHY

**Introduction to Control Hierarchy**

Control hierarchy  is also  called as the program structure. It basically  represents  the  hierarchy of control. It describes hierarchy of control by omitting  the procedural aspect which  covers  sequence of processes  occurrence or order of decisions  or repetition of  operations.

There are many different  notations which are use to represent control hierarchy the  most common is the  tree like  structure as shown in fig ,

Depth and width  provide an indication of the  number of levels of control and overall span of control, respectively. Fan  out a measure of the  number of modules that are directly controlled by another  module. fan in indicates how many modules directly control  a given module.



Program structure is usually expressed as a simple hierarchy showing super ordinate ( that  control another module ) and subordinate  9 that  is controlled by another module) relationships of modules. In fig 5.7, controller  module, M is super ordinate to  module a b and c whereas module f is subordinate of  module d and is ultimately subordinate to module M.

### [Software Design - Structural Partitioning](http://productdevelop.blogspot.com/2009/06/software-design-structural-partitioning.html)

The program structure should be partitioned both horizontally and vertically.  
  
(1)-g defines separate branches of the modular hierarchy for each major program function.  
  
Simplest way is to partition a system into: input, data transformation (processing), and output  
  
Advantages of horizontal partition:  
- easy to test, maintain, and extend  
- fewer side effects in change propagation or error propagation  
  
Disadvantage: more data to be passed across module interfaces  
- complicate the overall control of program flow  
  
(2) Vertical partitioning suggests the control and work should be distributed top-down in program structure.  
  
Advantages:  
- good at dealing with changes:  
- easy to maintain the changes  
- reduce the change impact and propagation

**Information hiding**

is a software design principle, where certain aspects of a program or module (the “secrets”) are inaccessible to clients. The primary goal is to prevent extensive modification to clients whenever the implementation details of a module or program are changed. This is done by hiding aspects of an implementation that might change behind a stable interface that protects clients from the implementation details. Users of that interface (whether it is a module, class, or function) will perform operations purely through its interface. This way, if the implementation changes, the clients do not have to change.

Information hiding serves as a criterion that can be used to [decompose a system into modules](https://embeddedartistry.com/fieldatlas/paper-on-the-criteria-to-be-used-in-decomposing-systems-into-modules/). The principle is also useful for reducing [coupling](https://embeddedartistry.com/fieldmanual-terms/coupling/) within a system.

# Effective Modular Design in Software Engineering

**The role of effective modular design in software engineering:**  
Any software comprises of many systems which contains several sub-systems and those sub-systems further contains their sub-systems. So, designing a complete system in one go comprising of each and every required functionality is a hectic work and the process can have many errors because of its vast size.

Thus in order to solve this problem the developing team breakdown the complete software into various modules. A module is defined as the unique and addressable components of the software which can be solved and modified independently without disturbing ( or affecting in very small amount ) other modules of the software. Thus every software design should follow modularity.

The process of breaking down a software into multiple independent modules where each module is developed separately is called **Modularization**.

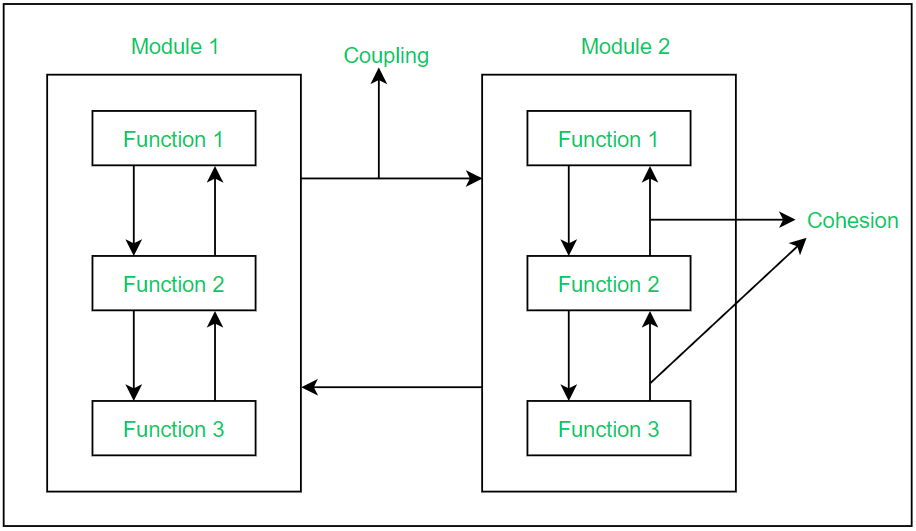
Effective modular design can be achieved if the partitioned modules are separately solvable, modifiable as well as compilable. Here separate compilable modules means that after making changes in a module there is no need of recompiling the whole software system.

In order to build a software with effective modular design there is a factor **“Functional Independence”** which comes into play. The meaning of Functional Independence is that a function is atomic in nature so that it performs only a single task of the software without or with least interaction with other modules. Functional Independence is considered as a sign of growth in modularity i.e., presence of larger functional independence results in a software system of good design and design further affects the quality of the software.

**Benefits of Independent modules/functions in a software design:**  
Since the functionality of the software have been broken down into atomic levels, thus developers get a clear requirement of each and every functions and hence designing of the software becomes easy and error free.

As the modules are independent they have limited or almost no dependency on other modules. So, making changes in a module without affecting the whole system is possible in this approach.  
Error propagation from one module to another and further in whole system can be neglected and it saves time during testing and debugging.

Independence of modules of a software system can be measured using 2 criteria : [Cohesion, and Coupling](https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/). These are explained as following below.



**Cohesion:**  
Cohesion is a measure of strength in relationship between various functions within a module. It is of 7 types which are listed below in the order of high to low cohesion:

**1.** Functional cohesion

**2.** Sequential cohesion

**3.** Communicational cohesion

**4.** Procedural cohesion

**5.** Temporal cohesion

**6.** Logical cohesion

**7.** Co-incidental cohesion

**Coupling:**  
Coupling is a measure of strength in relationship between various modules within a software. It is of 6 types which are listed below in the order of low to high coupling:

**1.** Data Coupling

**2.** Stamp Coupling

**3.** Control Coupling

**4.** External Coupling

**5.** Common Coupling

**6.** Content Coupling

*A good software design requires****high cohesion****and****low coupling****.*

### ***What is***functional independence in software engineering***?***

***Functional independence in software engineering*** means that when a module focuses on a single task, it should be able to accomplish it with very little interaction with other modules.

In software engineering, if a module is *functionally independent* of other module then it means it has **high cohesion** and **low coupling**.

*Functional independence* is essential for good software design.

# 