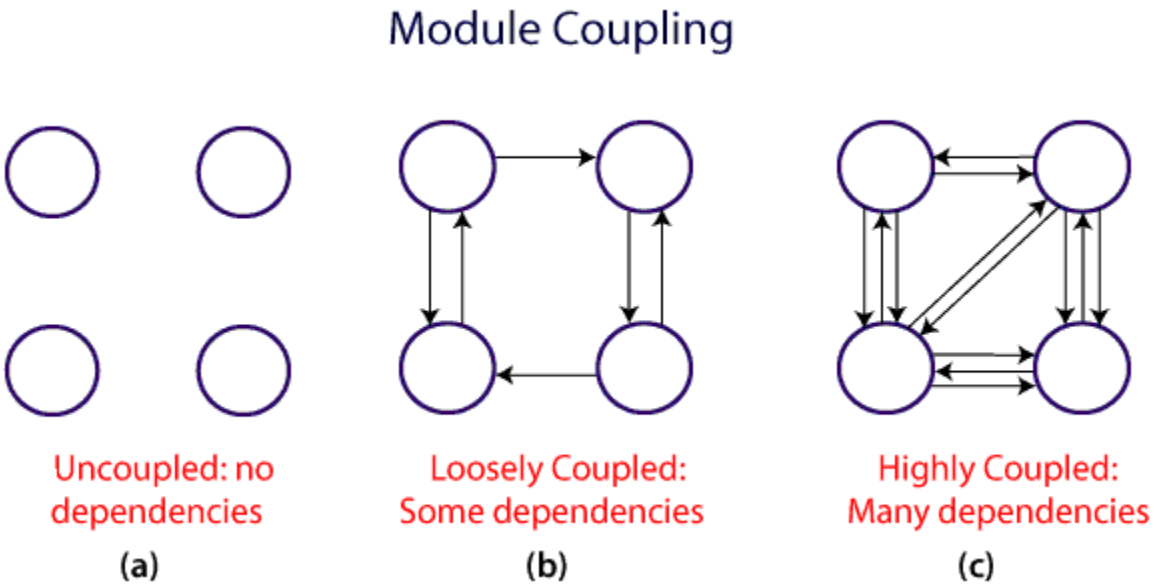
Coupling and Cohesion

Module Coupling

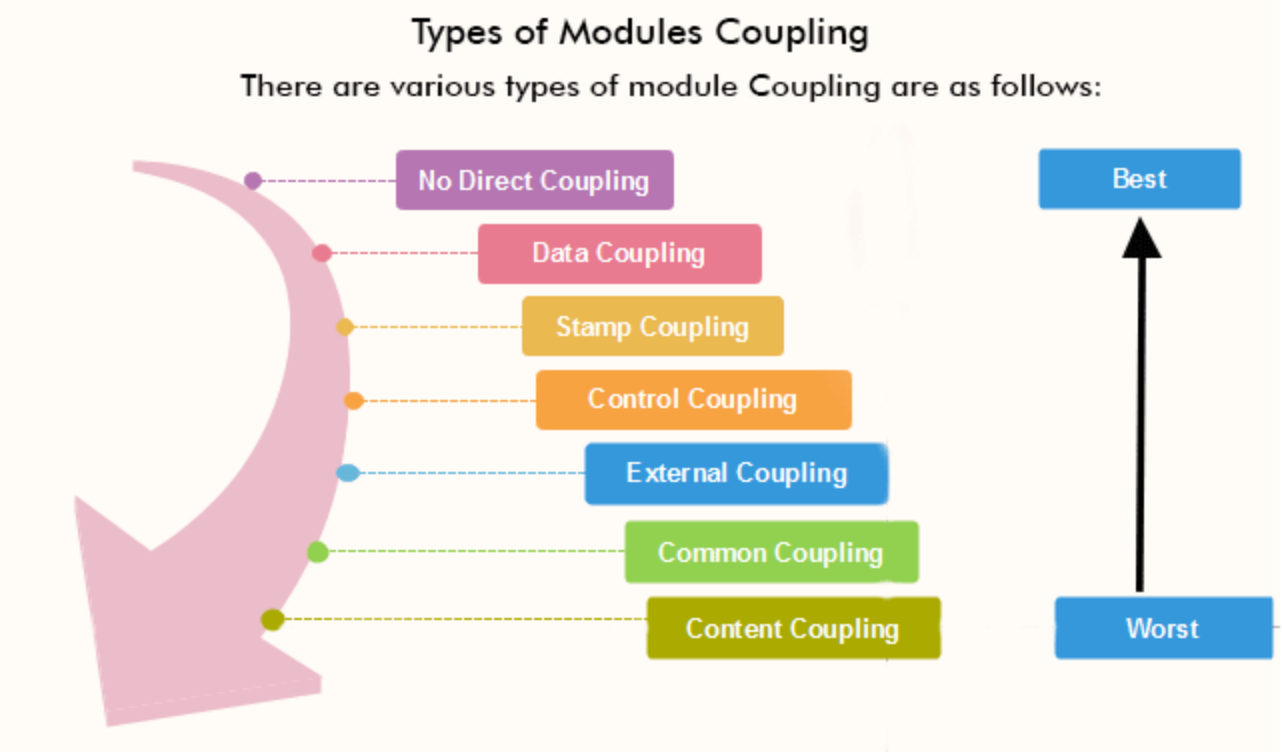
In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. **Uncoupled modules** have no interdependence at all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

The various types of coupling techniques are shown in fig:

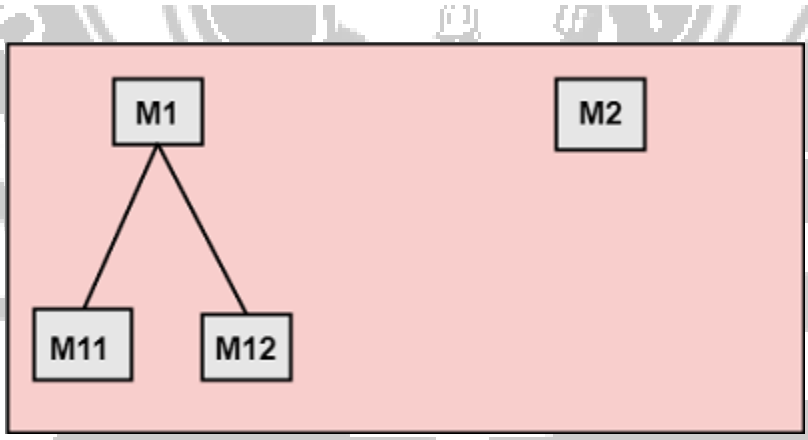


A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have more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

Types of Module Coupling

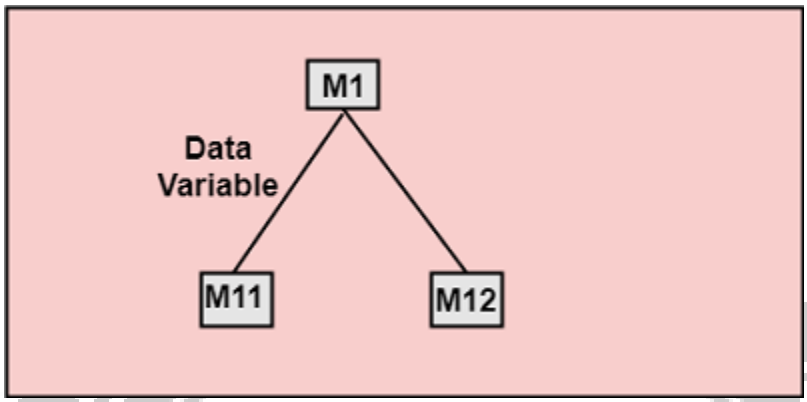


1. **No Direct Coupling:** There is no direct coupling between M1 and M2



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.

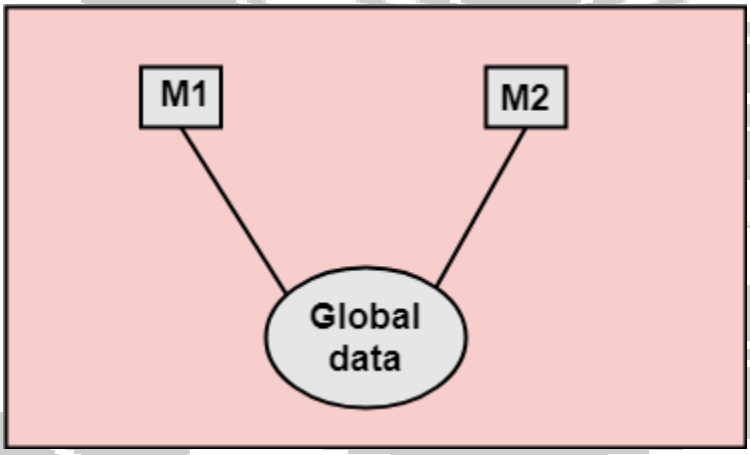


3. Stamp Coupling: Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object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

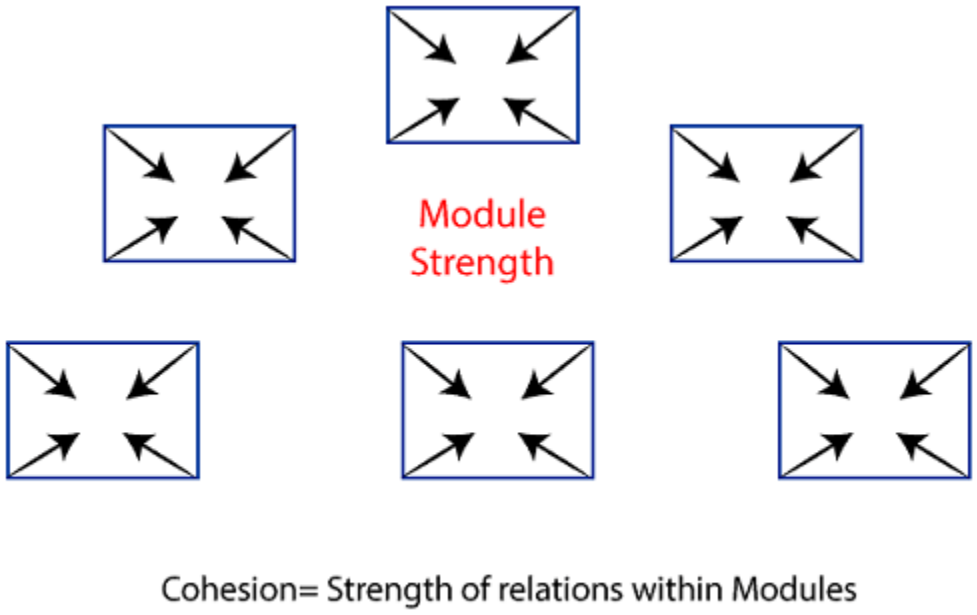


7. Content Coupling: Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

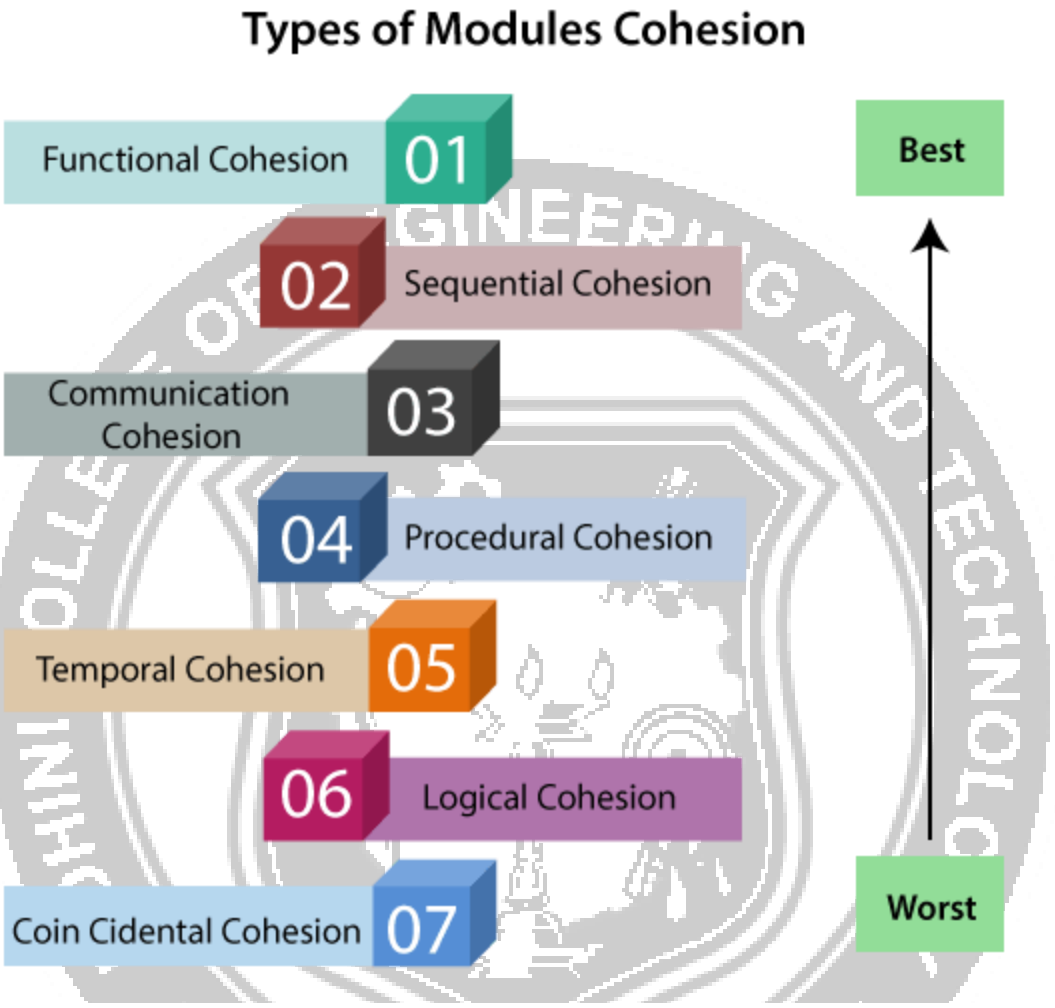
Module Cohesion

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."



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.

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.

Example of functional independence

We will take an example of simple college level project to *explain concept of functional independence*.

Suppose you and your friends are asked to work on a calculator project as a team. Here we need to develop each calculator functionality in form of modules taking two user inputs.

So our modules are addition Module, subtraction Module, division Module, multiplication Module. Each one of you pick up one module for development purpose.

Before you enter into development phase, you and your team needs to make sure to design the project in such a way that each of the module that you develop individually should be able to perform its assigned task without requiring much or no interaction with your friends module.

What I intend to say is, if you are working on addition Module then your module should be able to independently perform addition operation on receiving user input. It should not require to make any interaction with other modules like subtraction Module, multiplication Module or etc.

This is actually the concept of having module as *functional independence* of other modules. There is an advantage of functional independence in software engineering which we are going to discuss next.

Advantage of functional independence

Advantages of functional independence are given below:

Error isolation

When a module is functionally independent then it performs most of its task independently without interacting with other modules much. This reduces the chances of error getting propagated to other modules. This helps in easily isolating and tracing the error.

Module reusability

A functionally independent module performs some well defined and specific task. So it becomes easy to reuse such modules in different program requiring same functionality.

Understandability

A functionally independent module is less complex so easy to understand. Since such modules are less interaction with other modules so can be understood in isolation.