SE Principles

WEEK 8

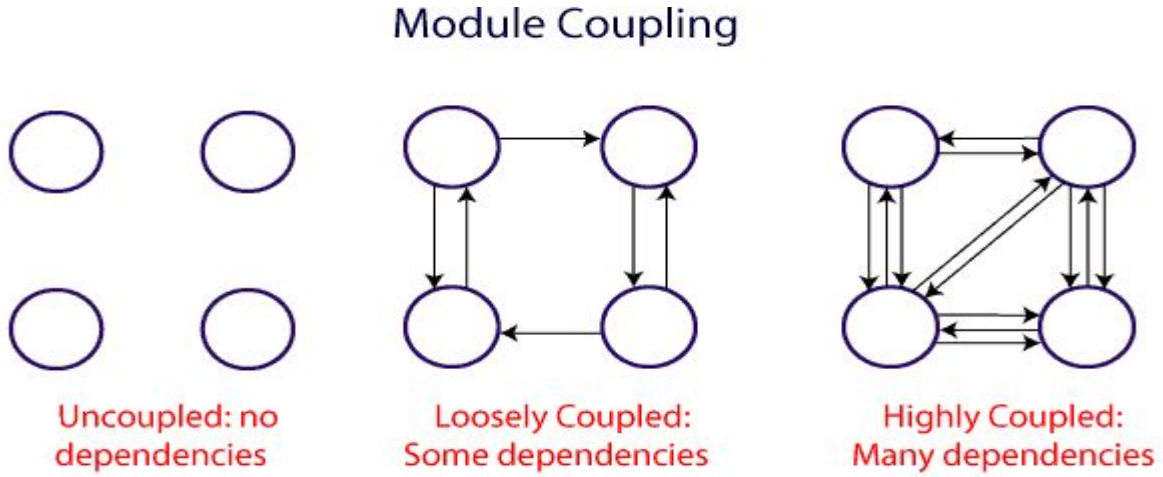
Coupling

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.



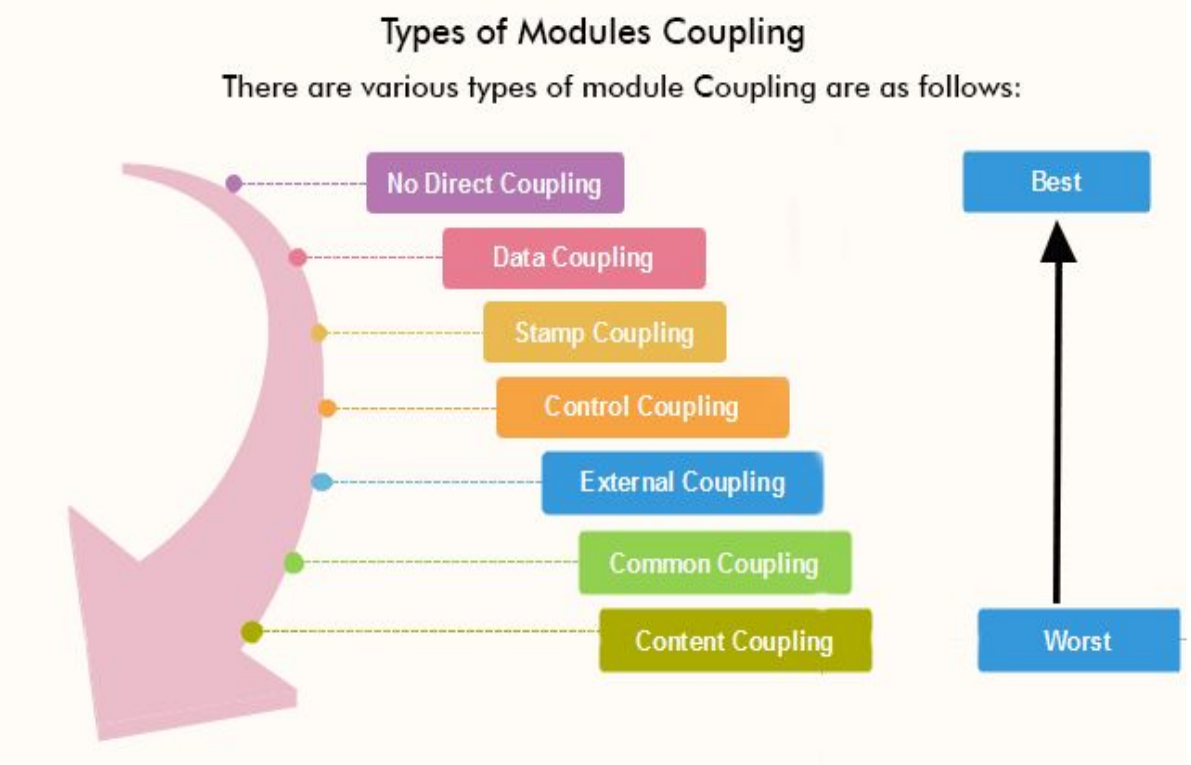
A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have more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

Types of Module Coupling

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

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.



Types of Module Coupling

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.

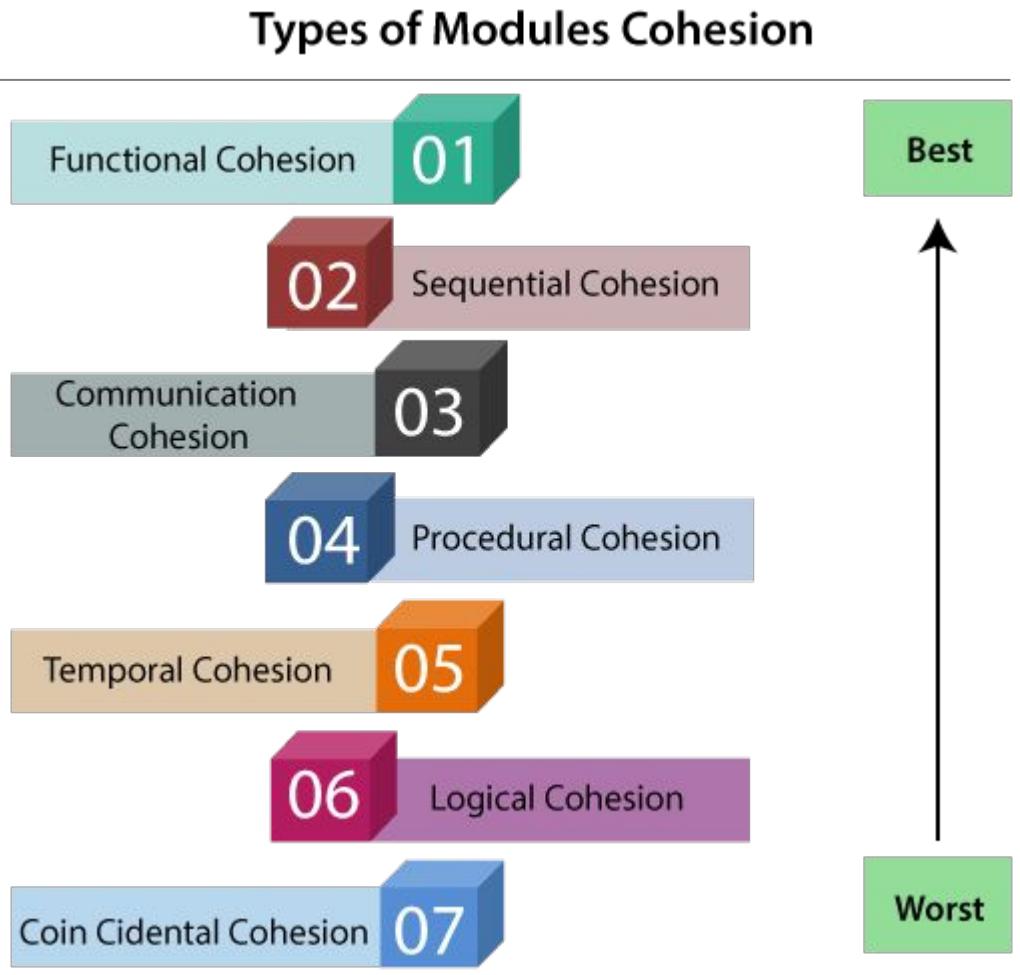
5. External Coupling: External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.

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.

Cohesion

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.



Types of Cohesion

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

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.

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.

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.

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.

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.

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.

Anticipation of Change

Constant change is inevitable in large software systems

- software repair and error elimination
- evolution of the application

Identify likely changes and plan for change

- software requirements usually not entirely understood
- users and environments change
- also affects management of software process

Maintenance is process of error correction and modification to reflect changing requirements

- regression testing with maintenance
- configuration management of versions

Anticipation of change helps to… …create a software infrastructure that absorbs changes easily …enhance reusability of components …control cost in the long run

Generality

Not generalizing often leads to continuous redevelopment of similar solutions

- Software development involves building many similar kinds of software (components)
- Software development cannot tolerate building the same thing over and over again

Generality leads to…

- …increased reusability
- …increased reliability
- …faster development
- …reduced cost

Incrementality

- It is not always possible in a software development project to know all the functional requirements in the beginning. With only partial requirements available, systems are often built on a piecemeal basis, incrementally adding on to the whole as more information is available.
- The principle of *incremental development* can be applied to situations where it is unclear how the functionalities of a system match up to the needs and expectations of customers. By developing an initial subset of the application for experimentation by customers, valuable feedback on the usefulness of the application can be gathered. Consequently, improvements based on the feedback are made to the application Through an iterative process of continuously seeking improvements to the application the final system can be produced.
- Incremental development is evolutionary