

**Functional Independence:**

The concept of *functional independence* is a direct outgrowth of modularity and the concepts of abstraction and information hiding. Functional independence is achieved by developing modules within with “signal-minded” function and an “aversion” to excessive interaction with other modules. Stated another way, we want to design software so that each module addresses a specific subfunction of requirements and has a simple interface when viewed from other parts of the program structure.

Independent modules are easier to develop because function may be compartmentalized and interfaces are simplified. Independence modules are easier to maintain (and test) because secondary effects caused by design\code modification are limited, error propagation is reduced. Independence is measured using two qualitative criteria: ***cohesion*** and ***coupling***. *Cohesion* is a measure of the relative functional strength of a module. *Coupling* is a measure of the relative interdependence among modules.

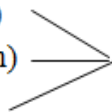
**COHESION** - the degree to which a module performs one and only one function.

**COUPLING** - the degree to which a module is "connected" to other modules in the system.

**Cohesion (binding)**

A SW component is said to exhibit a high degree of cohesion if the element in that unit exhibit a high degree of relatedness. This means that each element in the program unit should be essential for that unit to achieve its purpose. Seven levels of cohesion are identified in order of increasing strength of cohesion from lowest to highest.

1. ***Coincidental cohesion:*** the parts of a unit are not related but simply bundled together into a single unit.
2. ***Logical association:*** components which perform similar function such as input, error handling, etc. are put together in a single unit.

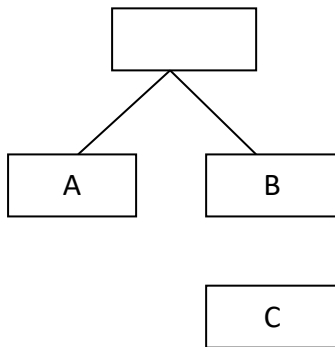
E.g. read from terminal (control message)  
Read from magnetic tape (transaction)  
Read from disk (record)       different formats

3. **Temporal cohesion:** all of the components which are activated at a single time are brought together.

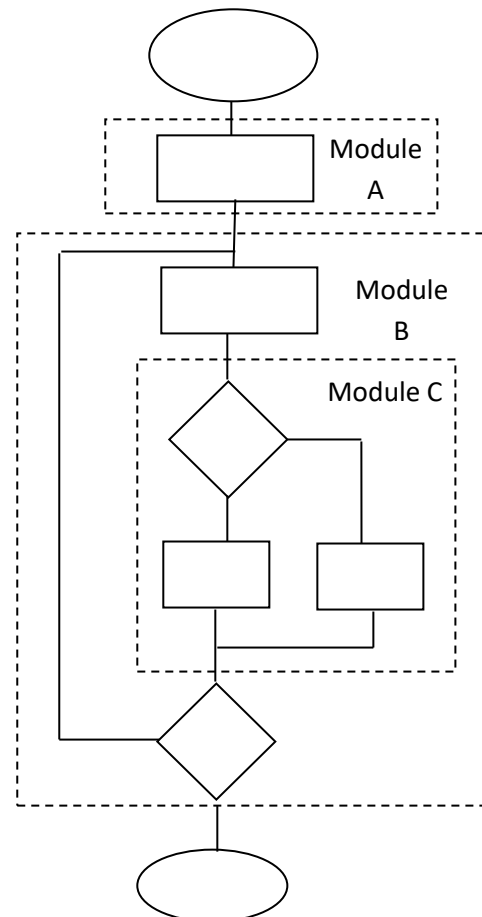
E.g. initialization module

- read control message
- open disk file
- reset counters

4. **Procedural cohesion:** the elements in a unit make up a single control sequence.



(Repetition, selection, seq)  
procedural constructs leads  
to procedural cohesion.



5. **Communicational cohesion:** all of the elements of a unit operate on the same input data or produce the same output data.
6. **Sequential cohesion:** the output from one element in the unit serves as input for some other element.
7. **Functional cohesion:** each part of the unit is necessary for the execution of a single function.

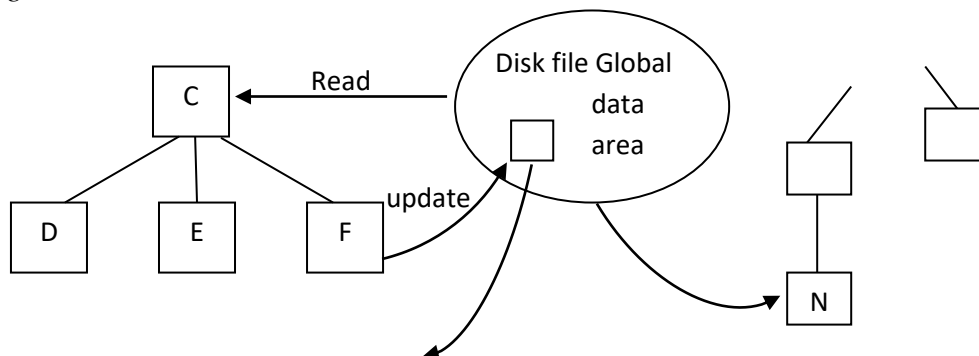
### **Coupling:**

Coupling is related to cohesion. It is an indication of the strength of interconnections between program units. Highly coupled systems have strong interconnections with program units

dependent on each other whereas loosely coupled systems are made up of units which are independent.

A number of coupling categories arises: (from high to low levels)

1. **Content coupling:** exists if one module refers to the inside of another (i.e. it branches into the other module).
2. **Common coupling:** exists if they refer to the same global data.  
E.g.



- How it is need to be changed?
  - How it is incorrectly updated by F for example?
  - SW will abort when N caller errors seem to be caused by N but the actual case is F, so diagnosis problems are difficult.
3. **Control coupling:** modules are interchanging control information. (E.g. control is passed via a flag on which decisions are made). When a module is passed an indicator telling the procedure which action to take from among a number of available actions (switch)  
E.g. general purposes of i/o procedure (command, device, buffer, length) commands has value 0,1,2,3...  
0: read, 1: write, 2: open...etc. this is undesirable because it is complicated. It can be split into several each carrying on a single action e.g. read device, buffer, and length).
  4. **Stamp coupling:** two modules are stamp coupled if they accept the same record data type as a parameter.
  5. **Data coupling:** exists if two modules communicate by parameters each one is a single data item which do not incorporate any control element.

**Effective Modular Design:**

A modular design reduces complexity, facilitates change (a critical aspect of software maintainability), and results in easier implementation by encouraging parallel development of different parts of a system. Abstraction and information hiding are used to define modules within software architecture.

***Module Types:-***

Within a program structure, a module may be categorized as:

1. *A sequential module* that is referenced and executed without a parent interruption by the application software.
2. *An incremental module* that can be interrupted prior to completion by application software and subsequently restarted at the point of interruption.
3. *A parallel module* that executes simultaneously with another module in concurrent multiprocessor environments.

***The conditions which must be considered in decomposition process in modules (properties of good design):***

1. **Module closure:**

Every module should realize a task which forms a closed entity. The functions which are included in a module should constitute a logical unit (for example they should operate on the same data structure). It is dangerous and there for undesirable, when a particular design decision (algorithm or data structure) gets divided among various modules.
2. **Interface Minimality and Visibility:**

The interface between two modules is the set of all assumptions which they make about each other (for example, name, and type of data). Every program system should be decomposed so that the interfaces between modules are as simple as possible; that is there should be few parameters and can be explicitly specified.
3. **Testability:**

Each module should be constructed so that its correctness can be determined simply by consideration of the interface without knowledge of how it is embedded in the completed system.
4. **Freedom from interface:**

The decomposition should be guarantee that modules do not exert any internal influence upon another and that each module may be replaced by another which respects the original interface with no effect on the complete system.

5. Module size:

Each module should be concise. This cannot be defined in terms of the number of lines or pages of source. The rule “not longer than one page” leads to misunderstanding that the complexity of a program is determined by its length. Well structured programs may be certainly comprise a number of pages but nevertheless be concise and under stable.

6. Module coupling:

One important objectives of modularization is to create largely independent (that is loosely coupled) modules. The fewer connections there are between modules, the less the danger of error propagation. This also reduces the danger that the changes in module will implicate change in a number of others. Because modules communicate with one another, there must be some module coupling. However, this should be accomplished exclusively through procedures and with a minimum number of parameters which do not communicate through parameters but rather access a common data domain, are closely coupled, and the search for errors is thereby impeded. The result is poor modularization, but parameters too, when missed, can lead to a close module coupling when a parameter is used to influence the internal flow, that is to say the logic of a called module, the calling module must know how the logic of the called module is constructed. Such “control parameters” also indicate decomposition.

7. Module Cohesion:

This is a measure of the interrelatedness of the functions of the module. Good or reasonable modularization should strive for strong module cohesions.

Strong cohesion means a functional, a data-oriented or a sequential relationship:

- a. A functionally connected module: consist of functions which are all necessary and sufficient to solve a particular task. Functional connectedness, therefore, can be directly derived from the process of stepwise refinement.
- b. A data oriented module: consist of functions which all access the same data structures.
- c. A sequentially connected module: consist of functions which must be carried out successively, and whose results are the prerequisite for the functions which follow.

8. The import number of a module: (fan-out)

This is the number of modules imported by a module. A large import number is a possible indication that the module has to perceive too many co-ordination and decision tasks. There are too few levels of abstraction. A small import number is perhaps an occasion for checking whether or not the module should be further decomposed.

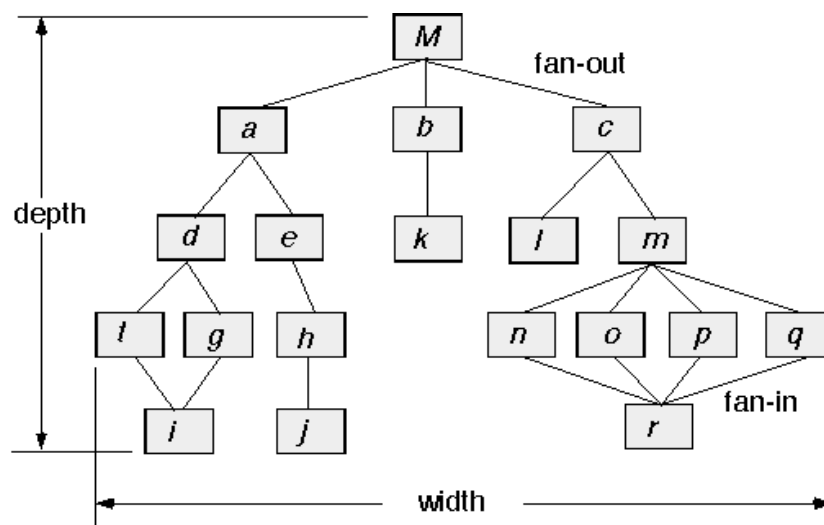
9. The utilization number of a module: (fan-in)

This is the number of modules by which a module is imported. The larger the utilization numbers of modules in a program system, the greater the ease of maintenance because only a small number of similar components must be processed during system service. But it must be guaranteed that such modules have strong module cohesion. It is not a trick to create a module with utilization number if that module is just a collection of arbitrary functions.

**10. Module hierarchy:**

Experience shows that the inner structures of a large number of program systems have a similar construction regardless of size or application. Looking at the finished design, four hierarchical levels of modules can be distinguished:

- Modules of the lowest level define data structures, data types, and access operations on individual elements of these data structures.
- The overlying level includes modules whose functions manipulate groups of elements of data structure. Thus both of these levels include functions which encapsulate data-oriented functions.
- The outermost modules contain control functions, which combine the functions of the lower levels in such a way that the system (specification) is satisfied.



**Program Architecture**