6.1.3 Modularity

The real power of partitioning comes if a system is partitioned into modules. ***A system is considered modular if it consists of discreet components so that each component can be implemented separately, and a change to one component has minimal impact on other components.*** Modularity helps in

- System debugging—isolating the system problem to a component is easier
- System repair—changing a part of the system is easy as it affects few other parts; and
- System building—a modular system can be easily built by "putting its modules together."

To achieve modularity, each module needs to support

- A well-defined abstraction and
- Interface through which it can interact with other modules.

6.1.4 Top-Down and Bottom-Up Strategies

A system is a hierarchy of components. The highest-level component correspond to the total system. To design such a hierarchy there are two possible approaches: top-down and bottom-up.

- **Top-down approach** starts from the highest-level component of the hierarchy and proceeds through to lower levels.
- **Bottom-up approach** starts with the lowest-level component of the hierarchy and proceeds through progressively higher levels to the top-level component.

a) Top-down design approach

It follows three steps

I. Identifying the major components of the system.
II. Decomposing them into their lower-level components.
III. Iterating until the desired level of detail is achieved.

This approach is a form of stepwise refinement. Starting from an abstract design to a more concrete level, until we reach a level where no more refinement is needed and the design can be implemented directly. Most design methodologies are based on the top-down approach.

A top-down approach is suitable only if the specifications of the system are clearly known and the system development is from scratch.

The components specified during design should be implementable, which requires some idea about the feasibility of the lower-level parts of a component.

b) Bottom-up approach

Bottom-up methods work with *layers of abstraction.* Starting from the very bottom, operations that provide a layer of abstraction are implemented. The operations of this layer are then used to implement more powerful operations, until the stage is reached where the operations supported by the layer are those desired by the system.

If a system is to be built from an existing system, a bottom-up approach is more suitable.

For a bottom-up approach to be successful, we must have a good notion of the top level. Without a good idea about the operations needed at the higher layers, it is difficult to determine what operations the current layer should support.

6.2 Module-Level Concepts

A module can be a macro, a function, a procedure (or subroutine), a process, or a package. To produce modular designs, some criteria must be used to select modules so that the modules support well-defined abstractions and are solvable and modifiable separately. In a system using functional abstraction, **coupling and cohesion** are two modularization criteria, which are often used together.

6.2.1 Coupling

Two modules are considered independent if one can function completely without the presence of other. However, all the modules in a system cannot be independent of each other, as they must interact so that together they produce the desired external behavior of the system.

Coupling between modules is the strength of interconnections between modules or a measure of interdependence among modules. **"Highly coupled" modules are joined by strong interconnections, while "loosely coupled" modules have weak interconnections. Independent modules have no interconnections.** To solve and modify a module separately, modules should be loosely coupled.

Constraint--The modules of the software system are created during system design, the coupling between modules is largely decided during system design and cannot be reduced during implementation.

To keep coupling low

- **Minimize the number of interfaces per module**. Coupling is reduced if only the defined entry interface of a module is used by other modules (for example, passing information to and from a module exclusively through parameters).
- **Minimize complexity of each interface.** Complexity of the entry interface of a procedure depends on the number of items being passed as parameters.
- **Type of information flow along the interfaces.** There are two kinds of information that can flow along an interface: **data or control.** Passing or receiving control information means that the action of the module will depend on this control information, which makes it more difficult to understand the module and provide its abstraction. Transfer of data information means that a module passes as input some data to another module and gets in return some data as output.

	Interface Complexity	Type of Connection	Type of Communication
Low	Simple obvious	To module by name	Data
			Control
High	Complicated obscure	To internal elements	Hybrid

Table 6.1: Factors affecting coupling.

Interfaces with only data communication result in the lowest degree of coupling, followed by interfaces that only transfer control data. Coupling is considered highest if the data is hybrid, that is, some data items and some control items are passed between modules.

6.2.2 Cohesion