UNIT-5

Module Specification:-

module specification A precise statement of the effects that a software module is required to achieve. It can be employed both by the implementer of the module, since it gives a definitive statement of the requirements that are imposed on the module, and by users of the module, since it gives a precise statement of what the module provides. A good module specification makes no commitment as to how the module's effects are achieved.

A variety of techniques have been developed for module specification. A *functional specification* identifies the operations that the module makes available and provides an individual specification for each operation, typically in the form of an input-output specification describing the mapping that the operation provides from a set of input values to a set of output values. In the typical case where a module has local data, a simple functional specification will need to refer to this local data when specifying each individual operation. This tends to obscure the specification, and also violates the principle that a specification should state what a module does but not how this is done.

The state machine model technique developed by Parnas treats the module as a finite-state machine and distinguishes operations that can observe the state of the machine from those that can alter the state of the machine. The specification is given by indicating the effect of each operation that can change the state on the result of each operation than can observe the state. This technique therefore avoids the need to refer to the module's local data.

The same applies to the technique of algebraic specification, largely due to Guttag and Horning. With this technique, which is tailored to the specification of abstract data type modules, the specification is given in two parts – a syntactic specification and a set of equations. The syntactic specification states the names, domains, and ranges of the operations provided by the module. Each equation specifies the net effect of some sequence of operations (or perhaps a single operation), and the complete set of equations must be sufficient to specify the effects of all operations under all conditions.

Because of the need for precision, module specifications are best given in some formal *specification language*. A variety of such languages have been developed, many drawing heavily on first-order predicate calculus. Specific examples include the SPECIAL language of the HDM system, which adopts the finite-state machine approach, and the language used by the AFFIRM system, which employs algebraic specification techniques.

Top down and bottom up design module coupling:-

A **top**-**down** approach (also known as stepwise **design**) is essentially the breaking **down** of a system to gain insight into the sub-systems that make it up. In a **top**-**down** approach an overview of the system is formulated, specifying but not detailing any first-level subsystems.

The **top**-**down** strategy uses the **modular approach** to develop the **design** of a system. It is called so because it starts from the **top** or the highest-level module and moves towards the lowest level modules. In this technique, the highest-level module or main module for developing the software is identified

he **top**-**down** strategy uses the **modular approach** to develop the **design** of a system. It is called so because it starts from the **top** or the highest-level module and moves towards the lowest level modules. In this technique, the highest-level module or main module for developing the software is identified.

Cohesion is the indication of the relationship within the module. **Coupling** is the indication of the relationships **between** modules. ... **Cohesion** is a degree (quality) to which a / module focuses on a single thing. **Coupling** is a degree to which a component/module is connected to the other modules

Bottom-**up** vs **Top**-**down Processing**. ... **Bottom**-**up** refers to the way it is built **up** from the smallest pieces of sensory information. **Top**-**down processing**, on the other hand, refers to perception that is driven by cognition. Your brain applies what it knows and what it expects to perceive and fills **in the** blanks, so to speak.

A **bottom**-**up** process is characterized by an absence of higher level direction in sensory processing, whereas a **top**-**down** process is characterized by a high level of direction of sensory processing by more cognition, such as goals or targets

That is the reason why **top**-**down designs** are written in plain English. The concept driving a **top**-**down design** is to break up the task that a program executes into a very few extensive subtasks. The highest level is known as the main module, **top** level or level 0. At this point, the volume of subtasks must be small.

A **top-down approach** (also known as stepwise design and in some cases used as a synonym of decomposition) is essentially the breaking **down** of a **system** to gain insight into its compositional sub-**systems** in a reverse engineering fashion.

A large part of **procedural programming** involves **Modular Elements**. These **elements** allow you to separate classes, data and functions meaning we have separate modules rather than a large jumble of work. Due to this it is best to use modules in large pieces of code so we can find errors easier and fix them.