5.2.4 Program Structure

The program structure, also called as control hierarchy, represents the hierarchy of control without regard to the sequences of processing and decisions. Program structure is usually expressed as a simple hierarchy showing super-ordinate and subordinate relationships of modules. Notations like Warnier-Orrnotation, Jackson notation, tree-like diagrams etc., are used to represent control hierarchy. The tree-like diagram (as shown in following figure) is the most common structure followed, which represents hierarchy of modules. The terms like depth, width, Fan-in, Fan-out are usually associated in describing and measuring the program structure.

Depth refers to the number of levels of control.

Width refers to the overall span of control.

Fan-in indicates how many modules directly control a given module.

Fan-out is a measure of the number of modules that are directly controlled by another module.

The relationship between modules is said to be either super-ordinate or subordinate. A module that controls another module is said to be super-ordinate to it. And a module controlled by another is said to be subordinate to the controller.

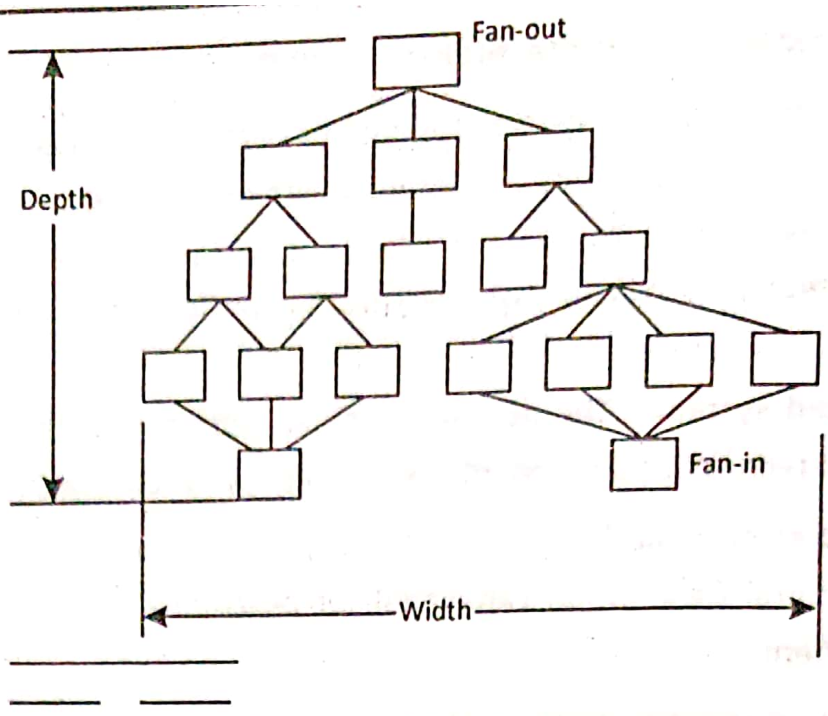

Fig. Control hierarchy

5.2.5 Data Structure

Data structure is a representation of the logical relationship among the individual elements of data. It represents the organization, methods of access, degree of associativity, and processing alternatives for problem-related information. Classic data structures include scalar, sequential, linked-list, n-dimensional, and hierarchical. Data structure, along with program structure, makes up the software architecture.

5.2.6 Modularity

A module is a named entity that:

(1) Contains instructions, processing logic, and data structures.

(2) Can be separately compiled and stored in a library.

(3) Can be included in a program.

(4) Module segments can be used by invoking a name and some parameters.

(5) Modules can use other modules.

Modularity derives from the architecture. Modularity is a logical partitioning of the software design that allows complex software to be manageable for purposes of implementation and maintenance. The logic of partitioning may be based on related functions, implementation considerations, data links, or other criteria. Modularity does imply interface overhead related to information exchange between modules and execution of modules. There are five important criteria for defining an effective modular system that enable us to evaluate a design method:

(1) Modular decomposability

If a design method provides a systematic way for decomposing a problem into sub problems, it will reduce the complexity of the overall problem, there by achieving an effective modular solution.