LAYERED ARRANGEMENT OF MODULES

- ***control hierarchy*** of a design is determined by the **order** in which different modules call each other. Many different types of notations have been used to represent the control hierarchy. The most common notation is a treelike diagram known as a *structure chart*.

- In a layered design solution, the modules are arranged into several layers based on their call relationships. A module is allowed to call only the modules that are at a lower layer. That is, a module should not call a module that is either at a higher layer or even in the same layer. Figure 4.1 (a) shows a layered design, whereas Figure 4.1 (b) shows a design that is not layered. Observe that the design solution shown in Figure 4.1(b), is actually not layered since all the modules can be considered to be in the same layer.

- In a layered design, the top-most module in the hierarchy can be considered as a manager that only invokes the services of the lower-level module to discharge its responsibility. The modules at the intermediate layers offer services to their higher layer by invoking the services of the lower layer modules and also by doing some work themselves to a limited extent. The modules at the lowest layer are the worker modules. These do not invoke services of any module and entirely carry out their responsibilities by themselves.

- In a layered design errors are isolated, since an error in one module can affect only the higher layer modules.

- In the following, we discuss some important concepts associated with a layered design:
  - *Superordinate and subordinate modules*: In a control hierarchy, a module that controls another module is said to be superordinate to it. Conversely, a module controlled by another module is said to be subordinate to the controller.

  - *Control abstraction:* In a layered design, a module should only invoke the functions of the modules that are in the layer immediately below it. In other words, the modules at the higher layers, should not be visible to the modules at the lower layers. This is referred to as control abstraction.

  - *Depth and width:* Depth and width of a control hierarchy provide an indication of the number of layers. For the design of Figure 4.1(a), the depth is 3 and width is also 3.

  - *Fan-out:* is a measure of the number of modules that are directly controlled by a given module. In Figure 4.1(a), the fan-out of the module M1 is 3. A design in which the modules have very high fan-out numbers is not a good design. The reason for this is that a very high fan-out is an indication that the module lacks cohesion and implement several different functions and not just a single cohesive function.

  - *Fan-in:* indicates the number of modules that directly invoke a given module. High fan-in represents code reuse and is in general, desirable in a good design. In Figure 4.1 (a), the fan-in of the module M1 is 0, that of M2 is 1, and that of M5 is 2.

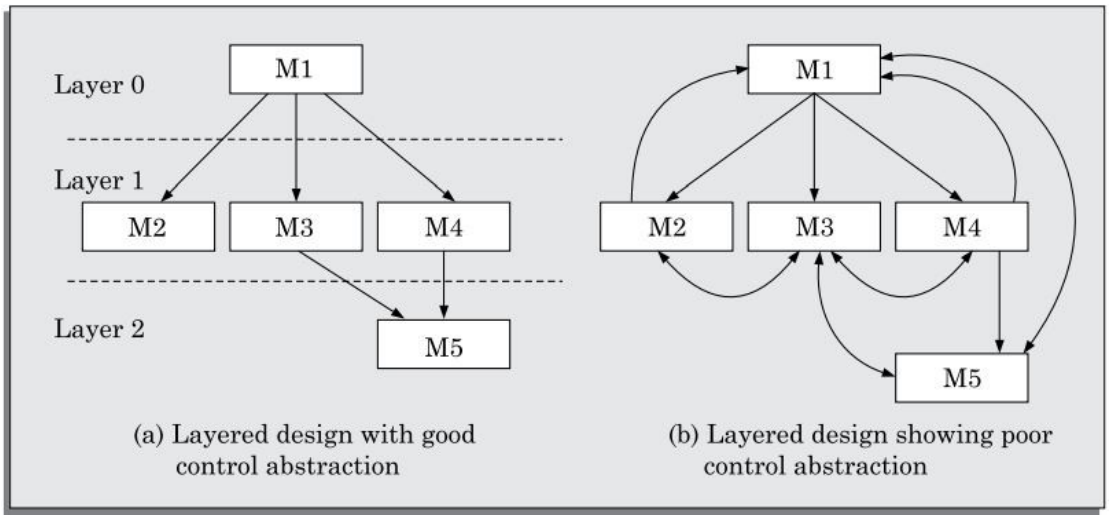

Figure 4.1: Examples of good and poor control abstraction

Structural Partitioning

If the architectural style of a system is hierarchical, the program structure can be partitioned both horizontally and vertically. Referring to Figure 4.2(a), horizontal partitioning defines separate branches of the modular hierarchy for each major program function. Control modules, represented in a darker shade are used to coordinate communication between and execution of the functions. The simplest approach to horizontal partitioning defines three partitions: input, data transformation (often called processing) and output. Partitioning the architecture horizontally provides a number of distinct benefits:

• software that is easier to test.

• software that is easier to maintain.

• propagation of fewer side effects.

• software that is easier to extend.

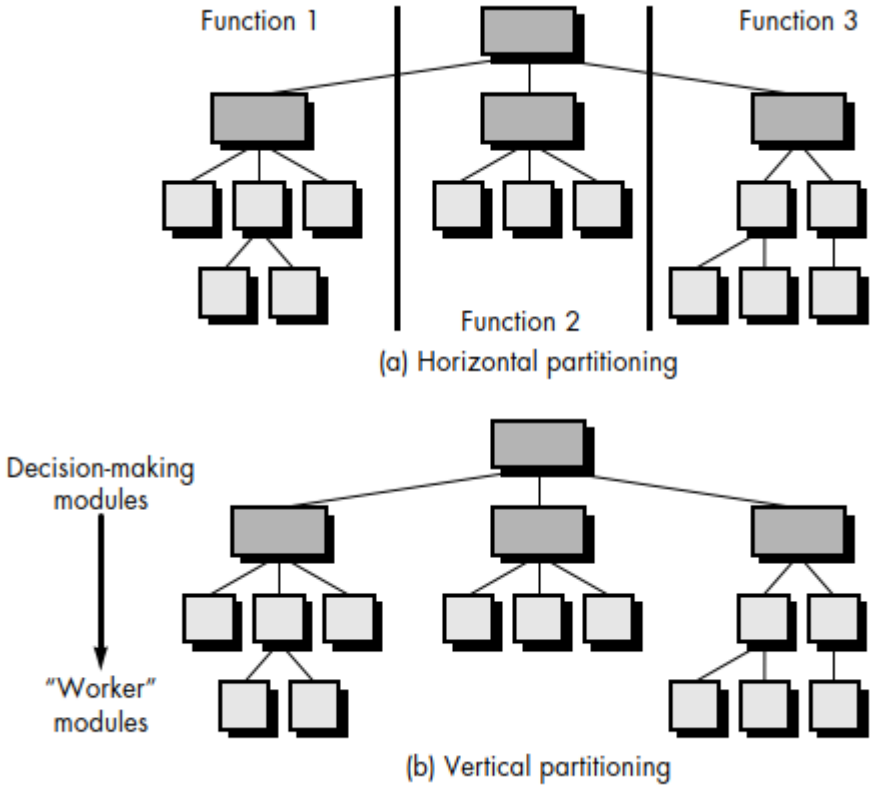

Figure 4.2: Structural partitioning

Vertical partitioning Figure 4.2(b), suggests that control and work should be distributed top-down in the program structure. Top level modules should perform control functions and do-little actual processing work.

Modules that reside low in the structure should be the workers, performing all input, computation, and output tasks.