# MODULARITY

## INTRODUCTION

In software engineering, modularity refers to the extent to which a software/Web application may be divided into smaller modules. Software modularity indicates that the application modules are capable of serving a specified business domain.

## COHESION

Cohesion as the "single-mindedness" of a component. Within the context of component-level design for object-oriented systems, cohesion implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself.

**Functional:** Exhibited primarily by operations, this level of cohesion occurs when a component performs a targeted computation and then returns a result.

**Layer:** Exhibited by packages, components, and classes, this type of cohesion occurs when a higher layer accesses the services of a lower layer, but lower layers do not access higher layers. Consider, for example, the SafeHome security function requirement to make an outgoing phone call if an alarm is sensed. It might be possible to define a set of layered packages. The shaded packages contain infrastructure components. Access is from the control panel package downward.

**Communicational:** All operations that access the same data are defined within one class. In general, such classes focus solely on the data in question, accessing and storing it. Classes and components that exhibit functional, layer, and communicational cohesion are relatively easy to implement, test, and maintain. You should strive to achieve these levels of cohesion whenever possible. It is important to note, however, that pragmatic design and implementation issues sometimes force you to opt for lower levels of cohesion.

## COUPLING

Communication and collaboration are essential elements of any object-oriented system. There is, however, a darker side to this important (and necessary) characteristic. As the amount of communication and collaboration increases (i.e., as the degree of "connectedness" between classes increases), the complexity of the system also increases. And as complexity increases, the difficulty of implementing, testing, and maintaining software grows.

Coupling is a qualitative measure of the degree to which classes are connected to one another. As classes (and components) become more interdependent, coupling increases. An important objective in component-level design is to keep coupling as low as is possible.

Coupling can manifest itself in a variety of ways:

**Content Coupling:** Occurs when one component "surreptitiously modifies data that is internal to another component". This violates information hiding—a basic design concept.

**Common Coupling:** Occurs when a number of components all make use of a global variable. Although this is sometimes necessary (e.g., for establishing default values that are applicable throughout an application), common coupling can lead to uncontrolled error propagation and unforeseen side effects when changes are made.

**Control Coupling:** Occurs when operation A() invokes operation B() and passes a control flag to B. The control flag then "directs" logical flow within B. The problem with this form of coupling is that an unrelated change in B can result in the necessity to change the meaning of the control flag that A passes. If this is overlooked, an error will result.

**Stamp Coupling:** Occurs when ClassB is declared as a type for an argument of an operation of ClassA. Because ClassB is now a part of the definition of ClassA, modifying the system becomes more complex.

**Data Coupling:** Occurs when operations pass long strings of data arguments. The "bandwidth" of communication between classes and components grows and the complexity of the interface increases. Testing and maintenance are more difficult.

**Routine Call Coupling:** Occurs when one operation invokes another. This level of coupling is common and is often quite necessary. However, it does increase the connectedness of a system.

**Type Use Coupling:** Occurs when component A uses a data type defined in component B (e.g., this occurs whenever "a class declares an instance variable or a local variable as having another class for its type"). If the type definition changes, every component that uses the definition must also change.

**Inclusion or Import Coupling:** Occurs when component A imports or includes a package or the content of component B.

**External Coupling:** Occurs when a component communicates or collaborates with infrastructure components (e.g., operating system functions, database capability, telecommunication functions). Although this type of coupling is necessary, it should be limited to a small number of components or classes within a system.