**5.1 The Design Process**

Software design is an iterative process through which requirements are translated- into a "blueprint" for constructing the software initially; the blueprint depicts a holistic view of software. That is, the design is represented at a high level of abstraction- a level that can be directly traced to the specific system objective and more detailed data, functional, and behavioral requirements. As design iterations occur, subsequent refinement leads to design representations at much lower levels of abstraction. These can still be traced to requirements, but the connection is more subtle.

Many design methods, are being applied throughout the industry. Like the analysis methods, each software design method introduces unique heuristics and notation, as well as a somewhat parochial view of what characterizes design quality. Yet, all of these methods have a number of common characteristics: (1) a mechanism for the translation of analysis model into a design representation, (2) a notation for representing functional components and their interfaces, (3) heuristics for refinement and partitioning, and (4) guidelines for quality assessment.

Software design is both a process and a model. The *design process* is a sequence of steps that enable the designer to describe all aspects of the software to be built. It is important to note, however, that the design process is not simply a cookbook. Creative skill, past experience, a sense of what makes "good" software and an overall commitment to quality are critical success factors for a competent design.

The *design model* is the equivalent of an architect's plans for a house. It begins by representing the totality of the thing to be built (e.g., a three-dimensional rendering of the house) and slowly refines the thing to provide guidance for constructing each detail (e.g., the plumbing layout). Similarly, the design model that is created for software provides a variety of different views of the computer software.

**5.2 Design Concept.**

A set of fundamental software design concepts has evolved over the past four decades. Although the degree of interest in each concept has varied over the years, each has stood the test of time. Each provides the software designer with a foundation from which more sophisticated, design methods can be applied.

**1. Abstraction**

There are three forms of abstraction. They are: (1) **procedural abstraction** is a named sequence of instructions that has a specific and limited function. An example of procedural would be the word open for a door. Open implies a long sequence of procedural steps (walk to the door, reach out and grasp knob, turn knob and pull door, step away from moving door, etc.), (2) **data abstraction** is a named collection of data that describes a data object. For the same example above, we can define a data abstraction called door because it encompass a set of attributes that describe the door, (3.) **control abstraction** is the third form of abstraction used in software design. Control abstraction implies a program control mechanism without specifying internal details. An example of a control abstraction is the synchronization semaphore used to coordinate activities in an operating system.

**2- Refinement**

Refinement is a top-down design strategy where a program is developed by successively refining levels of procedural detail. Abstraction and refinement are complementary concept. Abstraction enables a designer to specify procedure and data and yet suppress low-level details. Refinement helps the designer to reveal low-level details as design progresses. Both concepts aid the designer in creating a complete design model as the design evolves.

**3- Modularity**

Software is divided into separately named and addressable components, often called modules that are integrated to satisfy problem requirements. It has been stated that "modularity is the single attribute of software that allows program to be intellectually manageable".

There are-five criteria that used to evaluate a design method with respect to its ability to define an effective modular system:

1. **Modular decomposability.** If a design method provides a systematic mechanism for decomposing the problem into sub-problems, it will reduce the complexity of the overall problem, thereby achieving an effective modular solution.
2. **Modular composability**. If a design method enables existing design component to be assembled into a new system, it will yield a modular solution that does not reinvent the wheel.
3. **Modular understandability**. If a module can be understand as a standalone unit (without reference to other modules), it will be easier to build and easier to change.
4. **Modular continuity**. If small changes to the system requirements result in changes to individual module, rather than system wide changes.
5. **Modular protection.** If an aberrant condition occurs within a module and its effects are constrained within that module, the impact of error-induced side effects will be minimized.

**4. Software Architecture**

Software architecture alludes to the overall structure of the software and the ways in which that structure provides conceptual integrity for a system. Architecture is the hierarchical structure of program components (modules), the manner in which these components interact and the structure of data that are used by the components.

**5. Control Hierarchy**

Control hierarchy, also called program structure, represents the organization of program components (modules) and implies a hierarchy of control.

**6. Structure Partitioning**

If the architectural style of a system is hierarchical, the program structure can be partitioned both horizontally and vertically. Partitioning the architecture provides a number of distinct benefits: software that is easier to test, software that is easier to maintain, propagation of fewer side effect and software that is easier to extend.

**7. Data Structure**

Data structure is a representation of the logical relationship among individual elements of data. Because the structure of information will invariably affect the final procedural design, data structure is as important as program structure to the representation of software architecture.

**8. Software Procedure**

Software procedure focuses on the processing details of each module individually. Procedure must provide a precise specification of processing, including sequence of events, exact decision points, repetitive operations and even data organization and structure.

**9. Information Hiding**

Information hiding suggests that modules be characterized by design decisions that (each) hides from all other. In other words, modules should be specified and designed so that information (procedure and data) contained within a module is inaccessible to other modules that have no need for such information.