3.2 DESIGN CONCEPTS

Every intellectual discipline is characterized by fundamental concepts and specific techniques. Techniques are the manifestation of concepts as they apply to particular situations. Fundamental concepts of software design include

- Abstraction
- Software architecture
- Verification
- Structural Partitioning
- Object-Oriented Design Concepts
- Modularity
- Information hiding
- Control hierarchy
- Functional Independence
- Design Class
- Refinement
- Concurrency
- Date Structure
- Refactoring

3.2.1 Abstraction:

An abstraction is an intellectual tool that allows us to deal with concepts apart from particular instances of those concepts. It permits separation of conceptual aspects of the system from implementation details. In offering a modular solution to any problem, many levels of abstraction are possible such as

A higher level of abstraction: Solution is stated using the language of problem environment.

The lower level of abstraction: a Procedural approach which can be implemented.

Abstraction Mechanisms

There are three widely used abstraction mechanisms such as

1. **Functional Abstraction:** It involves the use of parameterized subprograms. These mechanisms allow us to control the complexity of the design process by systematically proceeding from abstract to concrete. (eg. packages in ADA, clusters in CLU)
2. **Data Abstraction:** It involves specifying a data type (or) data object by specifying the legal operations on objects. Many modern programming languages provide a mechanism for creating abstract data types which are used to denote declaration of data type (such as STACK, LIST)
3. **Control Abstraction**: Implies a program control mechanism without specifying internal details. An example of control abstraction is the synchronization semaphore used to coordinate activities in an operating system. Control abstraction permits specification of sequential subprograms, exception handlers, and coroutines without the exact details of implementation.

3.2.2 Modularity

The software is divided into separately named and addressable components often called modules that are integrated to satisfy problem requirements. Modularity helps in system debugging– isolating the system problem to a component is easier if the system is modular. For modularity, each module needs to support a well-defined abstraction and have a clear interface through which it can interact with other modules.

Modularity = ABSTRACTION + PARTITIONING

Desirable Properties of Modular System

1. Each processing abstraction is a well-defined subsystem that is potentially useful in other applications.
2. Each function in each abstraction has a single, well-defined purpose.
3. Each function manipulates no more than one major data structure.
4. Function share global data selectively.
5. Modularity enhances design clarity which in turn eases implementation, debugging, testing, documenting and maintenance.

Modularity is the single attribute of software that allows a program to be intellectually manageable. The number of control paths, a span of reference, number of variables and overall complexity would make understanding close to impossible. To illustrate this, consider the following argument based on observations of human problem-solving. Let C(x) be a function that defines the perceived complexity of problem x, and E(x) be a function that defines the effort (time) required to solve a problem x. For two problems P1 & P2 if

$$C(P1) > C(P2) \quad E(P1) > E(P2)$$

As a general case, this result is intuitively obvious and it takes more time to solve a difficult problem.

Another characteristic through experimentation is

$$C(P1 + P2) > C(P1) + C(P2)$$

(i.e.) the perceived complexity of a problem that combines P1 & P2 is greater than the perceived complexity where each problem is considered separately. Considering expression and the condition wiped by expressions, it follows that

$$E(P1 + P2) > E(P1) + E(P2)$$

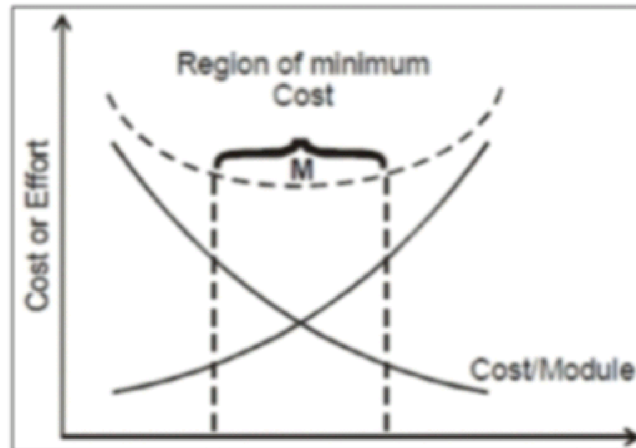

Fig. 3.1 Modularity and Software Cost

This leads to a "divide and conquers" conclusion - it's easier to solve a complex problem when it is broken into manageable pieces. Referring to Fig. 3.1, the effort (cost) to develop an individual software module does decreases as the total number of modules increases. Given the same set of requirements, more modules mean smaller individual size. However, as the number of modules increases, the effort associated with integrating the modules also grows. These characteristics lead to a total cost (or) effort curve shown in Fig. 3.1. There is a number M, of modules that would result in minimum development cost but we do not have the necessary sophistication to predict M with assurance. The curves provide useful guidance when modularity is considered. Modularization should be done with care to be taken to stay in the vicinity of M.

.Criteria for an effective modular system

1. Modular Decomposability: The design method should provide a systematic mechanism for decomposing the problem into subproblem to reduce the complexity of the overall problem.
2. Modular Composability: The design method enables existing (reusable) design components to be assembled into a new system.
3. Modular Under stability: The module should be easily under stable to build and charge.
4. Modular Continuity: If small changes to the system requirements result in changes to individual modules, rather than system-wide changes, the impact of change - induced side effects will be minimized.
5. Modular Protection: If an aberrant condition exists within a module and its effects are constrained within that module, the impact of error-induced side effects will be minimized.

3.2.3 Refinement

Stepwise refinement is a top-down design strategy in which a program is developed by successively refining levels of procedural detail. In each step, one (or) several instructions of the given program are decomposed into more detailed instructions. The refinement terminates when all instructions are expressed in terms of any underlying computer (or) programming language.

Refinement is a process of elaboration which causes the designer to elaborate on the original statement.

Abstraction and refinement are complementary concepts. 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 the design progresses.

3.2.4 Software Architecture

Architecture is the hierarchical structure of program components, the manner in which these components interact and structure of data of the components. One goal of software design is to derive an architectural rendering of a system.

3.2.6 Information Hiding

Information hiding is the fundamental design concept for software. The principle of information hiding suggests that modules be specified and designed so that the information (procedure and data) contained within a module is inaccessible to other modules that have no need for such information.

Elements for Information hiding

1. A data structure, its internal linkage and the implementation details of procedures that manipulate it (Data Abstraction)
2. The format of control blocks
3. Character codes, Ordering of character sets and other implementation details.
4. Shifting, masking and other machine dependent details.

The use of information hiding as design criteria provides benefits where modifications are required during testing (or) during software maintenance.

3.2.10 Functional Independence

Functional independence is the direct outgrowth of modularity and the concepts of abstraction and information hiding. Functional independence is achieved by developing modules with single-minded function with no interaction with other modules. Software with effective modularity (i.e.) independent modules is easier to develop because functions may be compartmentalized and interfaces are simplified. Independent modules facilitate

- Reduction in error propagation
- Reusability
- Easy maintenance and testing

Independence can be measured using cohesion and coupling.

3.2.11 Object-Oriented Design Concepts

The object-oriented (OO) paradigm is widely used in modern software engineering. These design concepts such as classes and objects, inheritance, messages, and polymorphism, among others are widely prevalent