Design Concepts

any 6 6m

- Abstraction
- Refinement
- Modularity
- Architecture
- Patterns
- Refactoring
- Functional Independence
- Information Hiding
- OO Design Concepts

Abstraction:-

It of two types 1) Lower Abstraction
2) Higher Abstraction

- At the highest level of abstraction, a solution is stated in broad terms. using the language of the problem environment. At lower levels of abstraction a more detailed description of the

solution is provided.

- As we move through different levels of abstraction, we work to create procedural and data abstraction
- A procedural abstraction refers to a sequence of instructions that have a specific and limited function. An example of a procedural abstraction would be the word "open" for a door
- A data abstraction is a named collection of data that describes a data object. In the context of the procedural abstraction open, we can define a data abstraction called "door"

Architecture

- A software Architecture defines the overall structure of software.
- The architecture consists of different model :-
  i) Structural models represent architecture as an organized collection of program
  ii) Framework models increase the level of design abstraction by attempting to identify repeatable architectural design frameworks that are encountered in similar types of applications.

iii) Dynamic models address the behavioral aspects of the program architecture, indicating how the structure or system configuration may change as a function of external events.

iv) Process models focus on the design of business or technical process that the system must accommodate.

v) Functional models:- can be used to represent the functional hierarchy of a system.

vi) Patterns:-

. A design pattern "conveys the essence of a proven design solution to a recurring problem within a certain context amidst computing concerns."

. The intent of each design pattern is to provide a description that enables a designer to determine.

1. whether the pattern is applicable to the current work,
2. whether the pattern can be reused, and
3. whether the pattern can serve as a guide for developing a similar, but functionally or structurally different pattern

vii) Modularity:-

. Software architecture and design

pattern embody modularity; that is, software is divided into separately named and addressable components, sometimes called modules that are integrated to satisfy problem requirements.

- Monolithic software (large program composed of a single module) cannot be easily grasped by a software engineer.
- It is the compartmentalization of data and function. It is easier to solve a complex problem when you break it into manageable pieces. "Divide-and-conquer".
- Don't over-modularize. The simplicity of each small module will be overshadowed by the complexity of integration "cost".

vii. Information Hiding:-

- It is about controlled interfaces. Modules should be specified and design so that information (algorithm and data) contained within a module is inaccessible to other modules that have no need for such information.
- The use of Information Hiding as a design criterion for modular systems provides the greatest benefits when modifications are required during testing

and later, during software maintenance.

- Because most data and procedures are hidden from other parts of the software, inadvertent errors introduced during modifications are less likely to propagate to other location within the software.

ix) Functional Independence:-

- Functional Independence is a key to good design and design is the key to software quality.
- Independence is assessed using two qualitative criteria: cohesion and coupling
- Cohesion is an indication of the relative functional strength of a module.
- Coupling is an indication of the relative inter-dependence among modules.
- Cohesive module should do just one thing
- Coupling is a qualitative indication of the degree to which a module is connected to other modules and to the outside world "lowest possible"

*) Refactoring:-

- It is reorganization technique that simplifies the design of a component without changing its function or behaviour
- When software is re-factored

the existing design is examined for redundancy, unused design elements, inefficie or unnecessary algorithms, poorly constructe data structures, or any other design failures that can be corrected to yield a better design.