Software Design

Introduction to design process

- The main aim of design engineering is to generate a model which shows firmness, delight and commodity.
- Software design is an iterative process through which requirements are translated into the blueprint for building the software.

Software quality guidelines

- A design is generated using the recognizable architectural styles and compose a good design characteristic of components and it is implemented in evolutionary manner for testing.
- A design of the software must be modular i.e the software must be logically partitioned into elements.
- In design, the representation of data, architecture, interface and components should be distinct.

Contd.,

- A design must carry appropriate data structure and recognizable data patterns.
- Design components must show the independent functional characteristic.
- A design creates an interface that reduce the complexity of connections between the components.
- A design must be derived using the repeatable method.
- The notations should be use in design which can effectively communicates its meaning.

Quality attributes

The attributes of design name as 'FURPS' are as follows:

Functionality:
It evaluates the feature set and capabilities of the program.

Usability:
It is accessed by considering the factors such as human factor, overall aesthetics, consistency and documentation.

Reliability:
It is evaluated by measuring parameters like frequency and security of failure, output result accuracy, the mean-time-to-failure(MTTF), recovery from failure and the program predictability.

Performance:

It is measured by considering processing speed, response time, resource consumption, throughput and efficiency.

Supportability:

- It combines the ability to extend the program, adaptability, serviceability. These three term defines the maintainability.
- Testability, compatibility and configurability are the terms using which a system can be easily installed and found the problem easily.
- Supportability also consists of more attributes such as compatibility, extensibility, fault tolerance, modularity, reusability, robustness, security, portability, scalability.

Design concepts

Fundamental software design concepts are as follows:

1. Abstraction

- A solution is stated in large terms using the language of the problem environment at the highest level abstraction.
- The lower level of abstraction provides a more detail description of the solution.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.
- A collection of data that describes a data object is a data abstraction.

- **Functional abstraction:** This involves the use of parameterized subprograms. Functional abstraction can be generalized as collections of subprograms referred to as 'groups'. Within these groups there exist routines which may be visible or hidden. Visible routines can be used within the containing groups as well as within other groups, whereas hidden routines are hidden from other groups and can be used within the containing group only.
- **Data abstraction:** This involves specifying data that describes a data object. For example, the data object *window* encompasses a set of attributes (window type, window dimension) that describe the window object clearly. In this abstraction mechanism, representation and manipulation details are ignored.
- **Control abstraction:** This states the desired effect, without stating the exact mechanism of control. For example, if and while statements in programming languages (like C and C++) are abstractions of machine code implementations, which involve conditional instructions. In the architectural design level, this abstraction mechanism permits specifications of sequential subprogram and exception handlers without the concern for exact details of implementation

2. Architecture

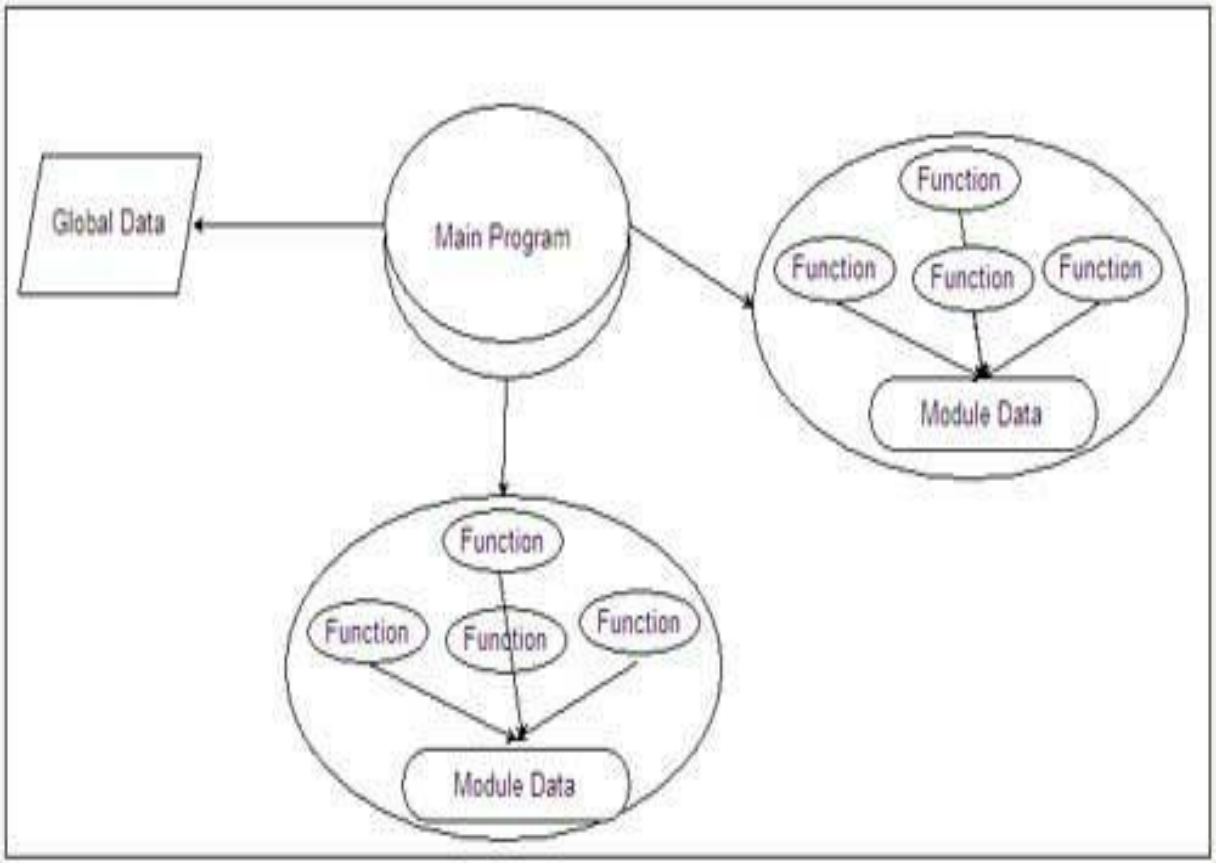
- The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.

3. Patterns

- A design pattern describes a design structure and that structure solves a particular design problem in a specified content.
- A pattern provides a description of the solution to a recurring design problem of some specific domain in such a way that the solution can be used again and again. The objective of each pattern is to provide an insight to a designer who can determine the following.
  - Whether the pattern can be reused
  - Whether the pattern is applicable to the current project
  - Whether the pattern can be used to develop a similar but functionally or structurally different design pattern.

4. Modularity

- A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.


Global Data
Main Program
Function
Function
Function
Function
Module Data
Function
Function
Function
Function
Module Data

Modularization

- Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently.
- These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.
- Modular design unintentionally follows the rules of ‘divide and conquer’ problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

Advantages of modularization

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again
- Concurrent execution can be made possible
- Desired from security aspect

Concurrency

- Back in time, all software are meant to be executed sequentially. By sequential execution we mean that the coded instruction will be executed one after another implying only one portion of program being activated at any given time.
- Say, a software has multiple modules, then only one of all the modules can be found active at any time of execution.
- In software design, concurrency is implemented by splitting the software into multiple independent units of execution, like modules and executing them in parallel.
- In other words, concurrency provides capability to the software to execute more than one part of code in parallel to each other.
- It is necessary for the programmers and designers to recognize those modules, which can be made parallel execution.

Example

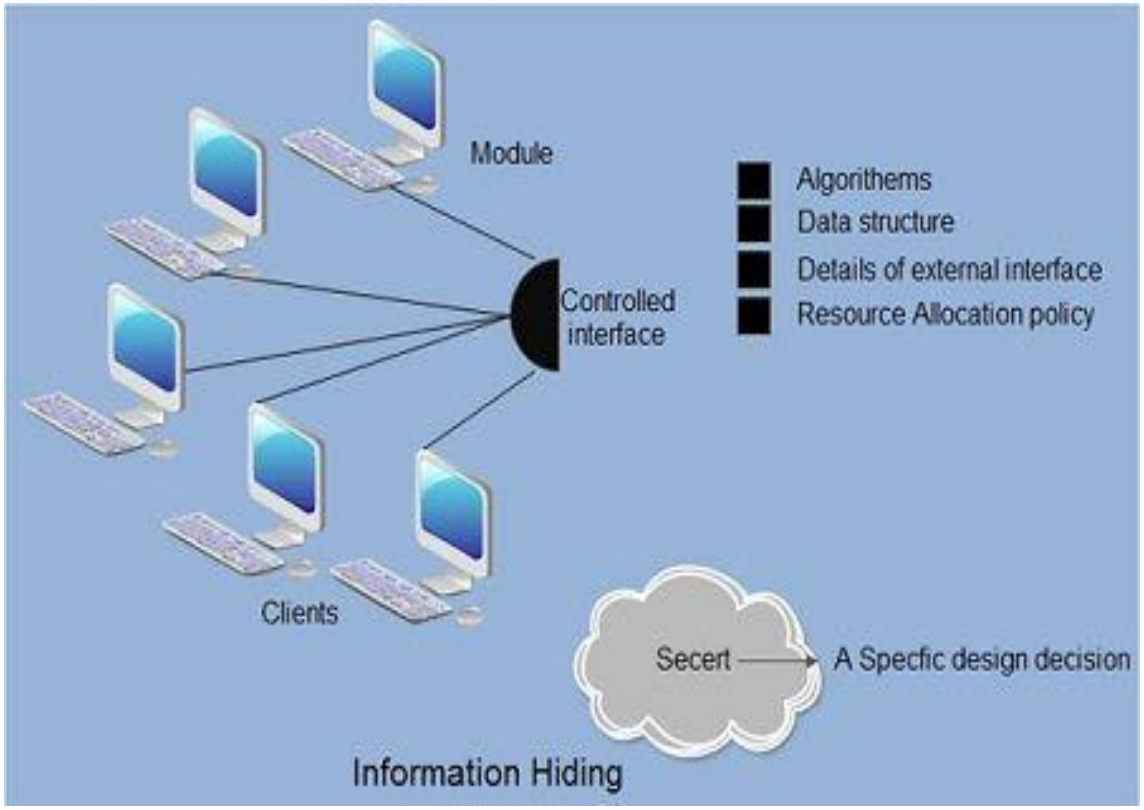
- The spell check feature in word processor is a module of software, which runs along side the word processor itself.

5. Information hiding

Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

6. Functional independence

- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- The functional independence is accessed using two criteria i.e Cohesion and coupling.



Some of the advantages associated with information hiding are listed below.

- Leads to low coupling
- Emphasizes communication through controlled interfaces
- Decreases the probability of adverse effects
- Restricts the effects of changes in one component on others
- Results in higher quality software.

Cohesion

- Cohesion is an extension of the information hiding concept.
- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

Coupling

- Coupling is an indication of interconnection between modules in a structure of software.

Coupling and Cohesion

- When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together.
- There are measures by which the quality of a design of modules and their interaction among them can be measured.
- These measures are called coupling and cohesion.

Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

- **Co-incidental cohesion -** It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.
- **Logical cohesion -** When logically categorized elements are put together into a module, it is called logical cohesion.
- **Temporal Cohesion -** When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.

- Cohesion: This is a measure of integrity and efficiency of a module. Unlike coupling this need not be a pairwise (relative to other modules) measure.
- The cohesion of a module is affected by the high coupling of its sub modules or its instructions. Suppose the 'solvequadratic' function internally computes the square roots it needs, its coupling quotient decreases but it cohesion is also reduced as the sqrt is not directly related to the solution process (it is not part of the algorithm).

They are mutual effectors.

- When cohesion is high so is coupling and when you try to reduce the dependancy of a module (make it more standalone) the cohesion automatically reduces.

- **Procedural cohesion -** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
- **Communicational cohesion -** When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- **Sequential cohesion -** When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
- **Functional cohesion -** It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

Coupling

- Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.
- There are five levels of coupling, namely -
- **Content coupling -** When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
- **Common coupling-** When multiple modules have read and write access to some global data, it is called common or global coupling.

- **Control coupling-** Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.
- **Stamp coupling-** When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
- **Data coupling-** Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

Ideally, no coupling is considered to be the best.

- The problem here is we have no proper system to measure coupling and cohesion, they are inherently qualitative terms. But if we are to measure them we can describe them as follows:
- Coupling: A measure of interdependency between program modules. It qualitatively measures how much stand alone a modules is with respect to other modules. Basically it is a pairwise metric. Suppose if I use the 'sqrt' function to calculate quadratic roots then I say they are somewhat coupled together.
- But if my 'solvequadratic' function can internally compute the solution numerically with out using the 'sqrt' or the 'surd roots' formula, then I can do away with coupling.

7. Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

8. Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- Refactoring is the process of changing the software system in a way that it does not change the external behaviour of the code still improves its internal structure.

9. Design classes

The model of software is defined as a set of design classes.

- Every class describes the elements of problem domain and that focus on features of the problem which are user visible.

- Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms.
- The output of this process can directly be used into implementation in programming languages.

- Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain.
- It tries to specify how to fulfill the requirements mentioned in SRS.

Software Design Levels

Software design yields three levels of results:

- **Architectural Design -** The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other.
- At this level, the designers get the idea of proposed solution domain.

- **High-level Design-** The high-level design breaks the ‘single entity-multiple component’ concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other.
- High-level design focuses on how the system along with all of its components can be implemented in forms of modules.
- It recognizes modular structure of each sub-system and their relation and interaction among each other.

- **Detailed Design-** Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs.
- It is more detailed towards modules and their implementations.
- It defines logical structure of each module and their interfaces to communicate with other modules.

Design Verification

- The output of software design process is design documentation, pseudo codes, detailed logic diagrams, process diagrams, and detailed description of all functional or non-functional requirements.
- The next phase, which is the implementation of software, depends on all outputs mentioned above.
- It is then becomes necessary to verify the output before proceeding to the next phase. The early any mistake is detected, the better it is or it might not be detected until testing of the product.

Contd.,

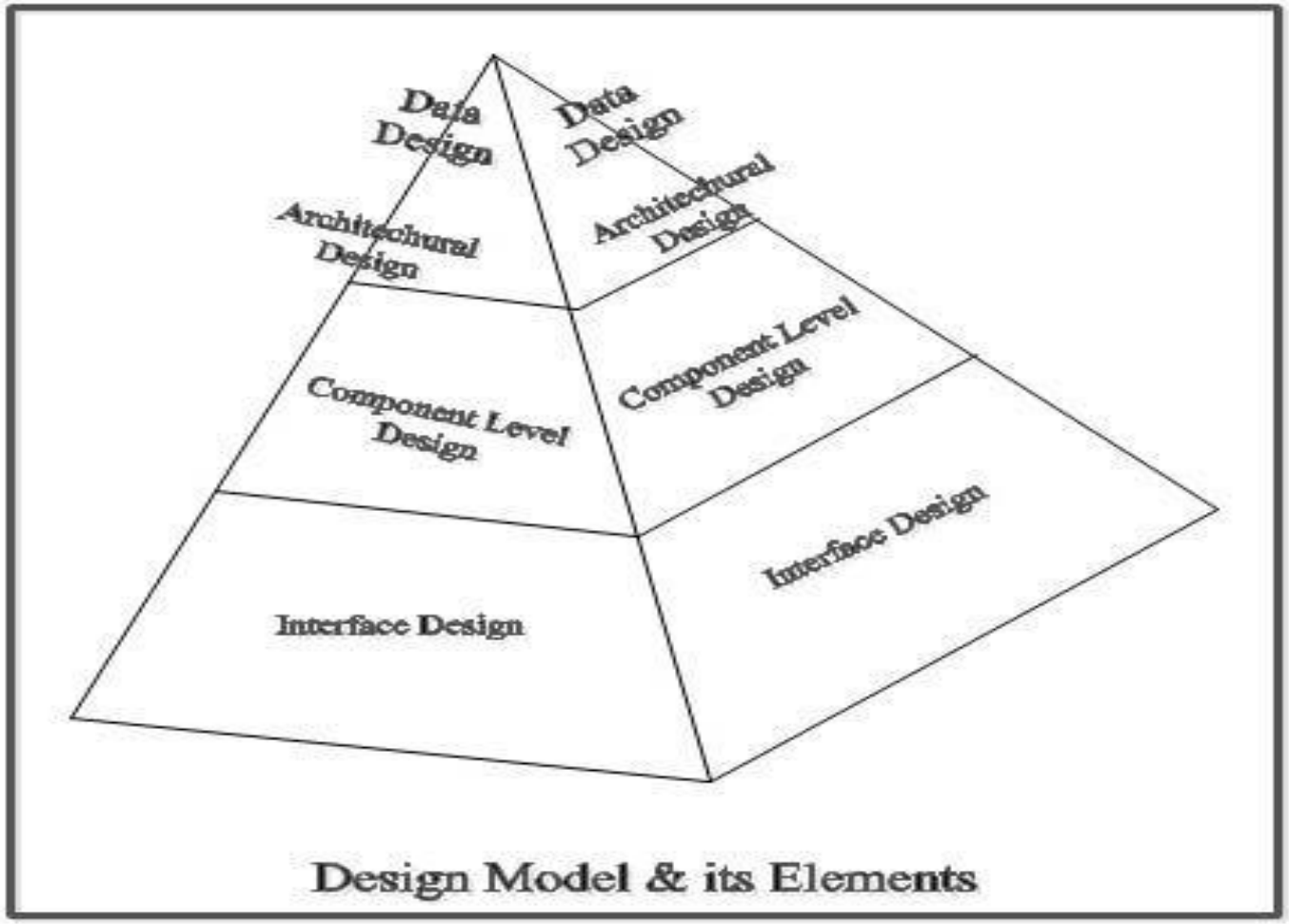
- If the outputs of design phase are in formal notation form, then their associated tools for verification should be used otherwise a thorough design review can be used for verification and validation.
- By structured verification approach, reviewers can detect defects that might be caused by overlooking some conditions.
- A good design review is important for good software design, accuracy and quality.

Developing a DesignModel

- To develop a complete specification of design (design model), four design models are needed. These models are listed below.
- **Data design:** This specifies the data structures for implementing the software by converting data objects and their relationships identified during the analysis phase. Various studies suggest that design engineering should begin with data design, since this design lays the foundation for all other design models.

Contd.,

- **Architectural design:** This specifies the relationship between the structural elements of the software, design patterns, architectural styles, and the factors affecting the ways in which architecture can be implemented.
- **Component-level design:** This provides the detailed description of how structural elements of software will actually be implemented.
- **Interface design:** This depicts how the software communicates with the system that interoperates with it and with the end-users.



Design Model & its Elements