# Chapter 4 :-
# Design Engineering
{

Prepared By :-
Jigar Dave
R K University
8469766496

Introduction to design process

- **The main aim of design engineering is to generate a model which shows determination, pleasure and product.**
- **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 separate.
- A design must carry appropriate data structure and identifiable data patterns.
- Design components must show the independent functional characteristic.
- A design creates an interface that reduce the complexity of connections between the components.
- The notations should be use in design which can effectively communicates its meaning.

Quality attributes

- **The attributes of design name as 'FRUPS' are as follows:**
  1) **F**unctionality:
  2) **R**eliability:
  3) **U**sability:
  4) **P**erformance:
  5) **S**upportability

- **1) Functionality:**
  It evaluates the feature set and capabilities of the program.

  2) **Usability:**
  It is accessed by considering the factors such as human factor,consistency and documentation.

  3) **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.

  4) **Performance:**
  It is measured by considering processing speed, response time, throughput and efficiency.

- **5) 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

- **1. Abstraction**
  - **A collection of data that describes a data object is a data 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.**

- **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.

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.
- **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.
- **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.

- **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.
- Functions refinement.

8.Refactoring

It is a reorganization technique which simplifies the design of components without changing its function behavior.

Refactoring is the process of changing the software system in a way that it does not change the external behavior of the code still improves its internal structure.

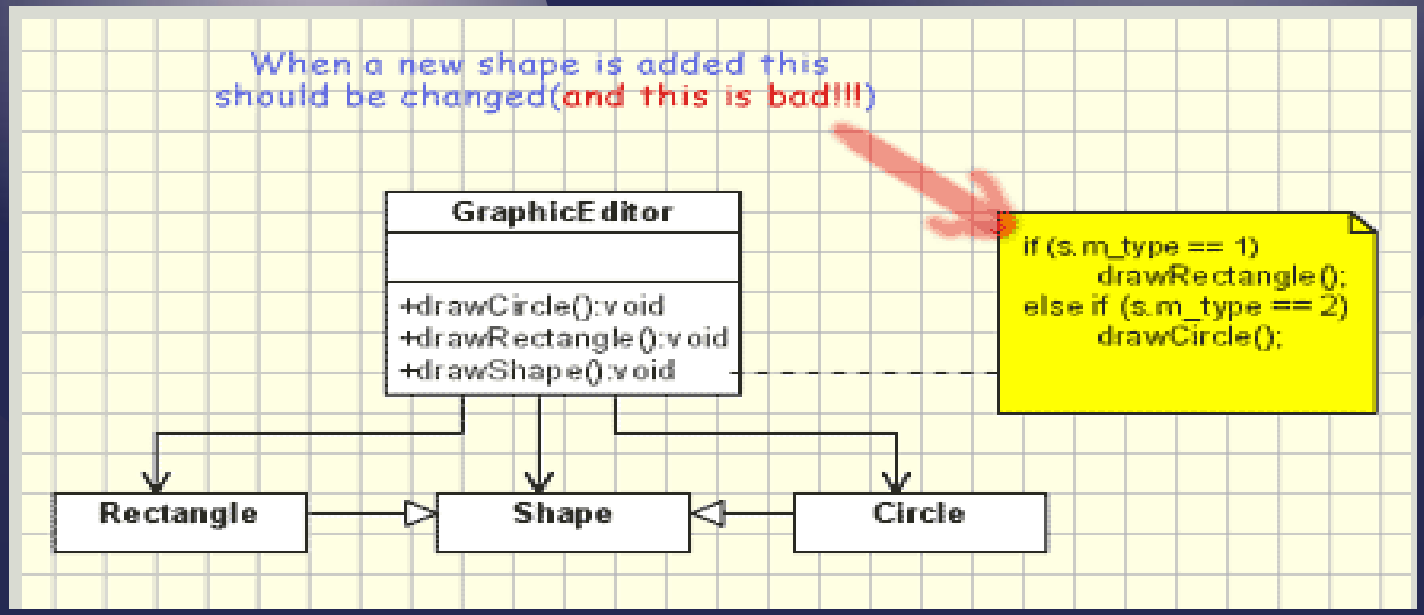
9. Design classes

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.

Open- closed Principle

- "Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification."
- The **Open Close Principle** states that the design and writing of the code should be done in a way that new functionality should be added with minimum changes in the existing code. The design should be done in a way to allow the adding of new functionality as new classes, keeping as much as possible existing code unchanged.



Structured Design

Function Oriented Design

- In function-oriented design, the system is comprised of many **smaller sub-systems known as functions**. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.
- Function oriented design inherits some properties of structured design where **divide and conquer** methodology is used.
- **Design Process :-**
- The whole system is seen as how data flows in the system by means of data flow diagram.
- DFD depicts how functions changes data and state of entire system.
- The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.
- Each function is then described at large.

Object Oriented Design

- Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics. The whole concept of software solution revolves around the engaged entities.
- **Objects -** All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on the attributes.
- **Classes -** A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object.
- In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.

- **Encapsulation -** In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.
- **Inheritance -** OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.
- **Polymorphism -** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

- **Design Process :-**
- Though it varies according to design approach (function oriented or object oriented, yet It may have the following steps involved:
- A solution design is created from requirement or previous used system and/or system sequence diagram.
- Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
- Class hierarchy and relation among them is defined.
- Application framework is defined.

Software Design Complexity

- Software design complexity is difficult to assess without using complexity metrics and measures.
- **Halstead's Complexity Measures :**
- In 1977, Mr. Maurice Howard Halstead introduced metrics to measure software complexity. Halstead's metrics depends upon the actual implementation of program and its actions, which are computed directly from the operators and operands from source code, in static manner. It allows to evaluate testing time, vocabulary, size, difficulty, errors, and efforts for C/C++/Java source code.

- He defines various indicators to check complexity of module.

Parameter	Meaning
n1	Number of unique operators
n2	Number of unique operands
N1	Number of total occurrence of operators
N2	Number of total occurrence of operands

When we select source file to view its complexity details in Metric Viewer, the following result is seen in Metric Report:

Metric	Meaning	Mathematical Representation
n	Vocabulary	n1 + n2
N	Size	N1 + N2
V	Volume	Length * Log2 Vocabulary
D	Difficulty	(n1/2) * (N1/n2)
E	Efforts	Difficulty * Volume
B	Errors	Volume / 3000
T	Testing time	Time = Efforts / S, where S=18 seconds.

```
main()
{
    int a, b, c, avg;
    scanf("%d %d %d", &a, &b, &c);
    avg = (a + b + c) / 3;
    printf("avg = %d", avg);
}
```

The unique operators are: `main`, `()`, `{}`, `int`, `scanf`, `&`, `=`, `+`, `/`, `printf`, `','`, `;`

The unique operands are: `a`, `b`, `c`, `avg`, `"%d %d %d"`, `3`, `"avg = %d"`

- $\eta_1 = 12, \eta_2 = 7, \eta = 19$
- $N_1 = 27, N_2 = 15, N = 42$
- Calculated Estimated Program Length: $\hat{N} = 12 \times log_2 12 + 7 \times log_2 7 = 62.67$
- Volume: $V = 42 \times log_2 19 = 178.4$
- Difficulty: $D = \frac{12}{2} \times \frac{15}{7} = 12.85$
- Effort: $E = 12.85 \times 178.4 = 2292.44$
- Time required to program: $T = \frac{2292.44}{18} = 127.357$ seconds
- Number of delivered bugs: $B = \frac{2292.44^{\frac{2}{3}}}{3000} = 0.05$

Cyclomatic Complexity Measures (C&C)

- Every program encompasses statements to execute in order to perform some task and other decision-making statements that decide, what statements need to be executed. These decision-making constructs change the **flow of the program.**
- If we compare two programs of same size, the one with more decision-making statements will be more complex as the control of program jumps frequently.
- McCabe, in 1976, proposed Cyclomatic Complexity Measure to quantify complexity of a given software. It is graph driven model that is based on decision-making constructs of program such as if-else, do-while, repeat-until, switch-case and goto statements.

- Process to make flow control graph :-
- Break program in smaller blocks, delimited by decision-making constructs.
- Create nodes representing each of these nodes.
- Connect nodes as follows:
  - If control can branch from block i to block j
  - Draw an arc
  - From exit node to entry node
  - Draw an arc.

- To calculate Cyclomatic complexity of a program module, we use the formula :-

- **$V(G) = e - n + 2$**

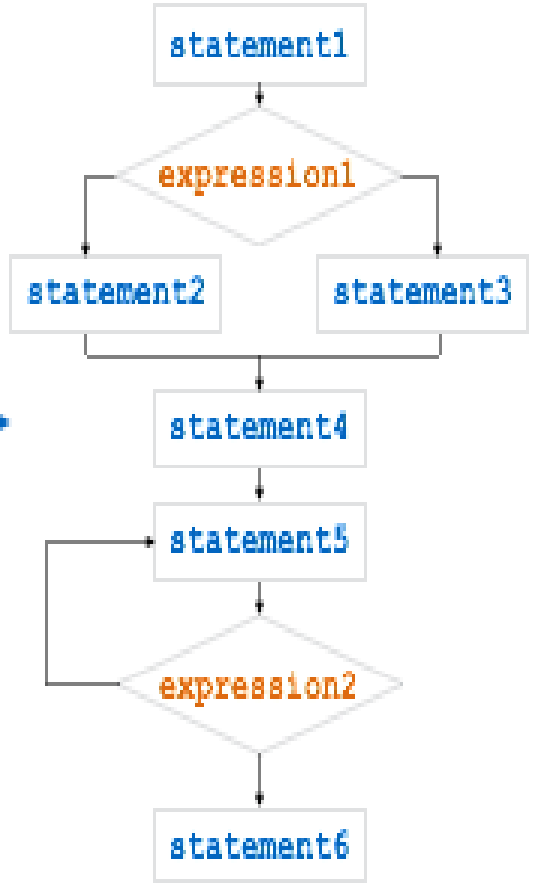
- Where
- e is total number of edges n is total number of nodes.
- n is total number of nodes.

- e = 10
- n = 8
- Cyclomatic Complexity = 10 - 8 + 2 = 4
- According to P. Jorgensen, Cyclomatic Complexity of a module should not exceed 10

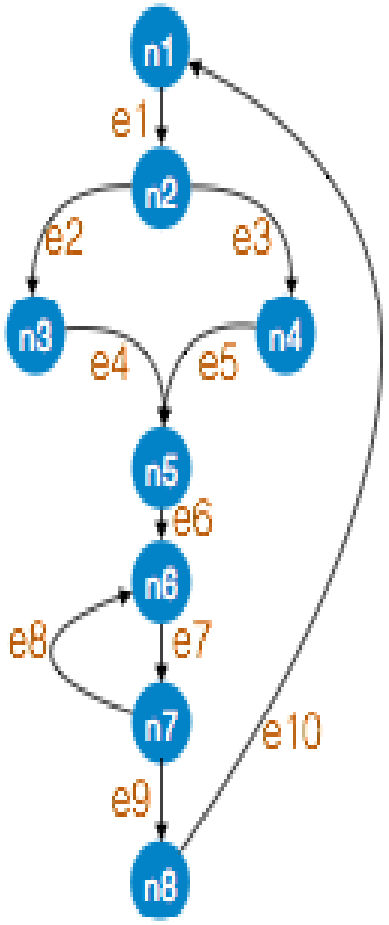
Code

```
statement1

If expression1
        statement2
else
        statement3

statement4

do
        statement5
while expression2

statement6
```

Flow-Chart



Flow-Graph



Role of Software Architecture

- The architecture of a system describes its major components, their relationships (structures), and how they interact with each other.
- Architecture serves as a **blueprint for a system**.
- It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.
- It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
- Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of −

- Selection of structural elements and their interfaces by which the system is composed.
- Behavior as specified in collaborations among those elements.
- Composition of these structural and behavioral elements into large subsystem.
- Architectural decisions align with business objectives.
- Architectural styles guide the organization.

- GOAL S FOR SOFTWARE ARCHITECTURE :-
- Expose the structure of the system, but hide its implementation details.
- Realize all the use-cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.
- Reduce the goal of ownership and improve the organization's market position.
- Improve quality and functionality offered by the system.
- Improve external confidence in either the organization or system.

The role of a software architect

- *interact with clients*
- *review the code*
- *collaborative working* (Client & Developer)

C&C View