**About Software Design Concepts**

* The software design concept simply means the idea or principle behind the design.
* It describes how you plan to solve the problem of designing software.
* It also shows the logic or thinking behind how you will design software.
* The software design concept for developing the right software provides a supporting and essential structure or model.

**Software Design Concepts**

1. Abstraction

2. Architecture

3. Design Patterns

4. Modularity

5. Information Hiding

6. Functional Independence

7. Refinement

8. Refactoring

9. Object-Oriented Design Concept

**1. Abstraction**

1. Abstraction is used to hide background details or unnecessary implementation about the data.
2. So that users see only required information.

**Type 1: Procedural Abstraction:**

* There is collections of subprograms.
* One is hidden group another is visible group of functionalities.

**Type 2: Data Abstraction:**

* Collections of data that describe data objects.
* Show representation data & hide manipulation data.
* **Examples:** Data Structure Programs directly used Push(), Pop(). Top() and Empty() methods.

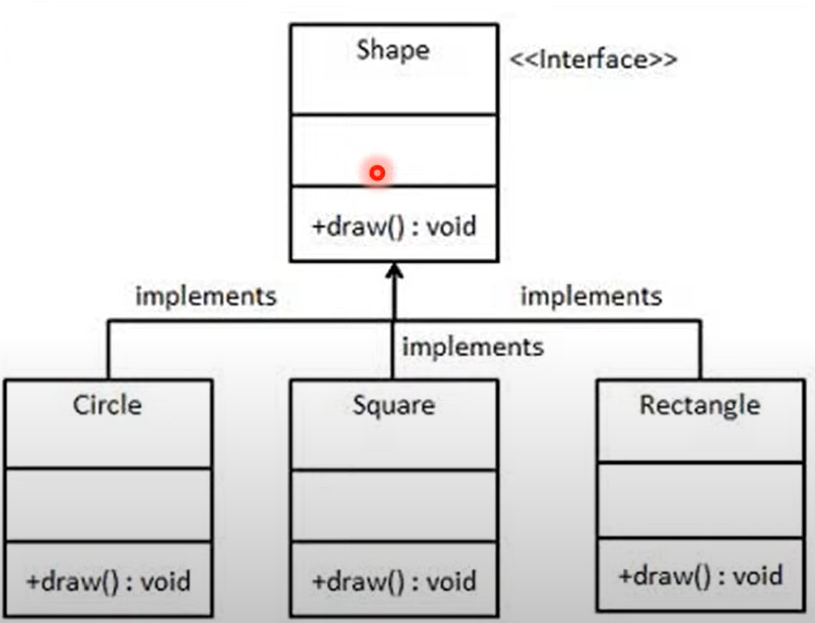
**2. Architecture**

• The architecture is the structure of program modules where they interact with each other in a specialized way.

* **Structural Properties:** Architectural design represent different types of components, modules, objects & relationship between these.
* **Extra-Functional Properties:** How design architecture achieve requirements of Performance, Capacity, Reliability, Security, Adaptability & other System Characteristics.
* **Families of related systems:** The architectural design should draw repeatable patterns. They have ability to reuse repeatable blocks.

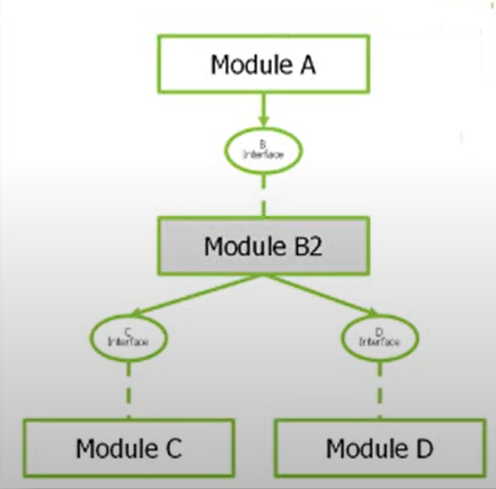
1. **Design Patterns**

* The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern.
* **Example:**



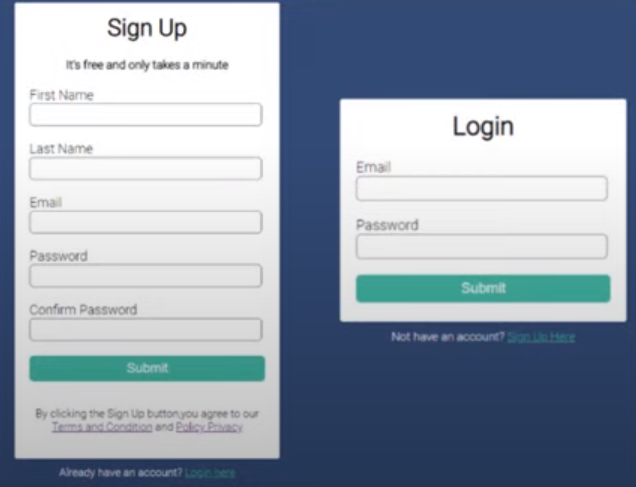
1. **Modularity**

* Modularity simply means dividing the system or project into smaller parts to reduce the complexity of the system or project.
* After developing the modules, they are integrated together to meet the software requirements.
* Modularizing a design helps to effective development, accommodate changes easily, conduct testing, debugging efficiently and conduct maintenance work easily.



1. **Information Hiding**

* Modules should be specified and designed in such a way that a data structures and algorithm details of one module are not accessible to other modules.
* They pass only that much information to each other, which is required to accomplish the software functions.
* The way of hiding unnecessary details in modules is reffered to as information hiding.



**6. Functional Independence**

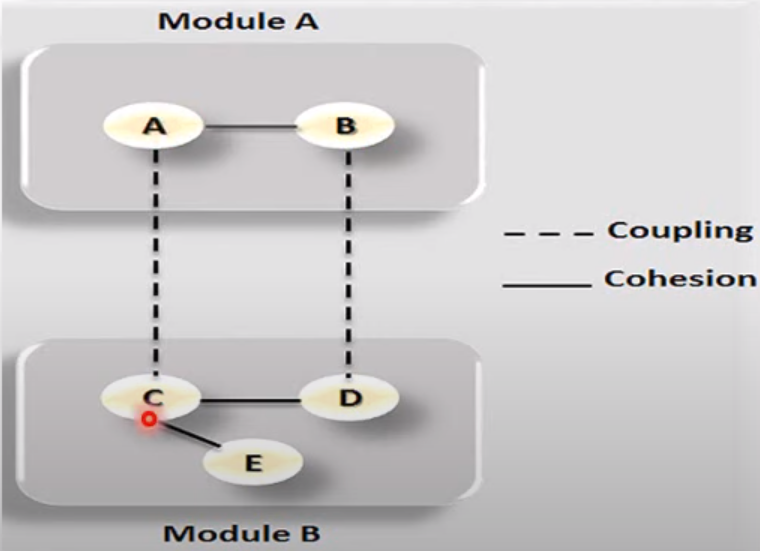
* The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.

**Criteria 1: Coupling**

* The degree in which module is "connected" to other module in the system.
* Low Coupling necessary in good software.

**Criteria 2: Cohesion**

* The degree in which module perform functions in inner module in the system.
* High Cohesion necessary in good software.



**7. Refinement**

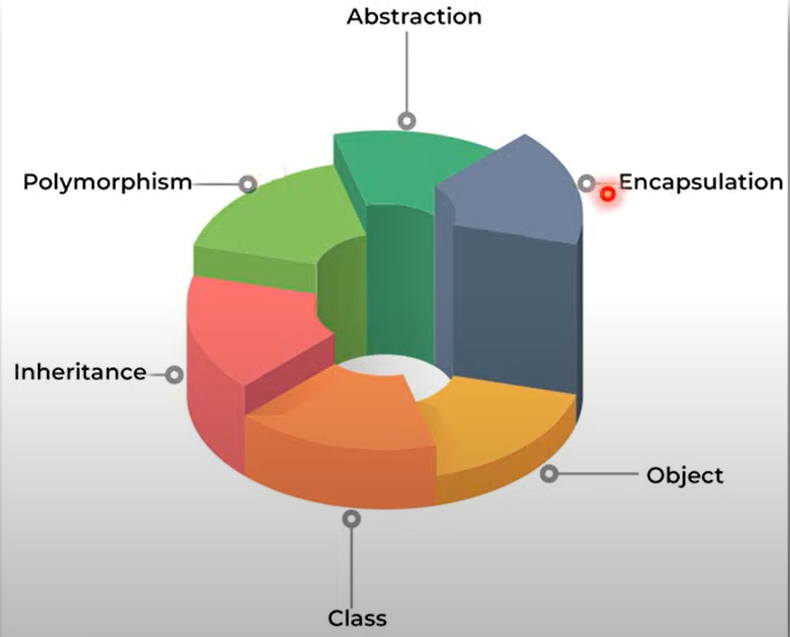
* Refinement is a top-down design approach.
* It is a process of elaboration.
* A program is established for refining levels of procedural details.
* A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

**8. Refactoring**

* Refactoring is the process of changing the internal software system in a way that it does not change the external behavior of the code still improves its internal structure.
* When software is refactored the existing design is examined for redundancy, unused design elements, unnecessary design algorithms, poorly constructed data, inappropriate data structure or any other design failure that can be corrected for better design.

**9.Object Oriented Design Concepts**

* Object Oriented is a popular design approach for analyzing and designing an application.
* Advantage is that faster, low cost development and creates a high quality software.



**Function-oriented design**

Function-oriented design is an approach to software engineering that focuses on designing software systems based on the functions or tasks they need to perform. It is also known as procedural design or procedural decomposition. In function-oriented design, the emphasis is on breaking down the problem into smaller sub-problems and designing functions or procedures to solve each sub-problem.

Here are some key characteristics and principles of function-oriented design:

**Modularity:** Function-oriented design promotes modularity by dividing a software system into small, manageable functions. Each function is responsible for performing a specific task or sub-task within the system. This modular structure enhances code reusability and maintainability.

**Top-down design**: Function-oriented design follows a top-down approach, starting with the main function or module and decomposing it into smaller functions. This process continues until the functions are small enough to be implemented directly.

**Procedural abstraction**: Function-oriented design utilizes procedural abstraction to encapsulate functionality within functions. Each function hides the details of its implementation and provides a well-defined interface for interaction with other functions.

**Sequential flow**: Function-oriented design typically follows a sequential flow of control, where functions are called in a specific order to achieve the desired outcome. Functions may receive inputs, process them, and produce outputs that serve as inputs for subsequent functions.

**Data sharing through parameters**: Function-oriented design relies on passing data between functions through parameters. Functions can receive inputs as parameters, perform operations on them, and return outputs as results or modified parameters.

**Focus on algorithms and logic**: Function-oriented design emphasizes the design of algorithms and logical flow to solve problems. It focuses on step-by-step procedures and control structures such as conditionals (if-else statements) and loops (for, while) to achieve the desired functionality.

**Limited code reuse**: Function-oriented design does not inherently promote code reuse to the same extent as other design paradigms like object-oriented design. However, by designing functions to be modular and independent, some code can still be reused across different parts of the system.

While function-oriented design has been widely used in the past, it is less prevalent in modern software engineering practices, where object-oriented and other design paradigms are more popular. However, function-oriented design can still be applicable and effective in certain scenarios, especially when dealing with smaller, task-specific programs or algorithmic implementations.

**Object oriented design**

Object-oriented design (OOD) is a popular approach to software engineering that focuses on designing software systems based on the concept of objects. In object-oriented design, the emphasis is on modeling real-world entities as objects that encapsulate data and behavior. It promotes code reusability, maintainability, and modularity by organizing the system into classes, objects, and their interactions.

Here are some key characteristics and principles of object-oriented design:

**Classes and objects**: Object-oriented design revolves around the concept of classes and objects. A class is a blueprint or template that defines the properties (data members) and behaviors (methods) that objects of that class possess. An object is an instance of a class, representing a specific entity or instance of the class.

**Encapsulation**: Encapsulation is a principle that promotes bundling data and related behaviors within an object, hiding the internal details from other objects. It ensures that the object's internal state is accessed and modified only through defined interfaces or methods, providing data protection and abstraction.

**Inheritance**: Inheritance allows the creation of new classes (derived or child classes) based on existing classes (base or parent classes). The derived classes inherit the properties and behaviors of the base class, enabling code reuse and the establishment of hierarchical relationships between classes.

**Polymorphism**: Polymorphism enables objects of different classes to be treated as instances of a common superclass. It allows methods to be defined in the superclass and overridden in the derived classes, providing flexibility and extensibility. Polymorphism facilitates the use of a single interface to represent multiple related classes.

**Abstraction**: Abstraction focuses on capturing the essential characteristics of an object or system while ignoring unnecessary details. It allows the creation of abstract classes and interfaces that define common properties and behaviors, which can be inherited and implemented by concrete classes.

**Association and composition**: Object-oriented design represents relationships between objects using association and composition. Association signifies a loosely coupled relationship between objects, where one object can reference another. Composition represents a stronger relationship where one object is composed of other objects, and the lifetime of the composed objects is controlled by the containing object.

**Message passing**: Object-oriented design promotes communication between objects through message passing. Objects interact with each other by invoking methods or sending messages to request or provide services. Message passing facilitates the collaboration and cooperation between objects to achieve desired functionalities.

**Design patterns**: Object-oriented design often utilizes design patterns, which are proven solutions to common software design problems. Design patterns provide reusable templates for designing classes, objects, and their interactions. Examples of design patterns include the Singleton pattern, Factory pattern, Observer pattern, and many more.

Object-oriented design has gained widespread adoption in modern software engineering due to its flexibility, modularity, and reusability. It supports the development of scalable and maintainable systems by promoting clean separation of concerns and modeling real-world entities in a natural way. Object-oriented programming languages like Java, C++, and Python provide built-in features and syntax to facilitate object-oriented design principles.

**Detailed Design**

Detailed design in software engineering refers to the process of transforming high-level design specifications into a more detailed and concrete representation of the software system. It involves making design decisions at a lower level of abstraction, defining the structure, components, and algorithms necessary for the implementation of the software.

Here are some key aspects and considerations in the detailed design phase:

**Component Design**: Detailed design focuses on defining the individual components or modules that make up the software system. This includes determining the internal structure of each component, specifying interfaces, and designing the data structures and algorithms required within each module.

**Data Design**: Detailed design involves designing the data structures and databases required for the software system. This includes defining the types of data to be stored, organizing and representing the data, and specifying the relationships between different data elements.

**Algorithm Design**: Detailed design involves designing the algorithms and procedures that will be used to perform specific tasks within the software system. This includes selecting appropriate algorithms, defining the step-by-step logic for performing operations, and optimizing algorithms for efficiency and performance.

**Interface Design**: Detailed design focuses on designing the interfaces between different components, modules, and external systems. This includes specifying the methods, parameters, and data formats required for communication and interaction between different software entities.

**Error Handling and Exception Handling**: Detailed design includes planning for error handling and exception handling within the software system. This involves identifying potential error scenarios, defining error handling mechanisms, and specifying how the system will handle exceptions and unexpected situations.

**Security and Performance Considerations**: Detailed design involves addressing security and performance concerns. This includes designing appropriate security mechanisms, such as authentication and encryption, and considering performance optimizations, such as efficient data structures and algorithms, caching strategies, and scalability considerations.

**Documentation**: Detailed design requires documenting the design decisions and rationale. This includes creating detailed design documents, diagrams, and specifications that provide a comprehensive understanding of the system's structure and behavior.

**Design Review and Validation**: Detailed design should undergo review and validation to ensure its correctness, completeness, and adherence to the high-level design specifications. This may involve peer reviews, code inspections, and architectural reviews to identify and address any design flaws or inconsistencies.

**Iterative and Incremental Refinement**: Detailed design is an iterative process that involves refining and improving the design through multiple iterations. As implementation progresses, new insights may arise, requiring design modifications and updates.

It's important to note that detailed design is closely connected to the preceding phases of requirements analysis and high-level design. It should align with the requirements and constraints identified during those phases while incorporating any changes or refinements made during the high-level design stage.

By performing a thorough and well-executed detailed design, software engineers can ensure that the software system is accurately and efficiently implemented according to the desired functionality, quality attributes, and architectural guidelines.

**Verification**

Verification and Validation is the process of investigating that a software system satisfies specifications and standards and it fulfills the required purpose. Barry Boehm described verification and validation as the following:

Verification: Are we building **the product right**?

Validation: Are we building the right product?

Verification:

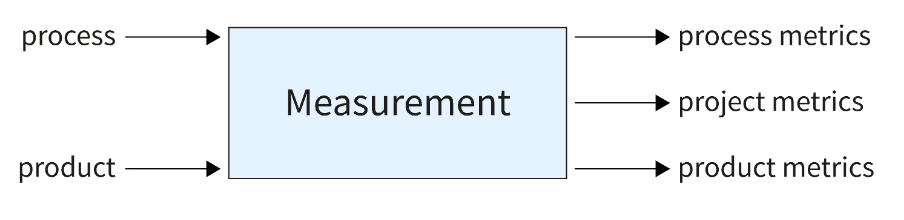
Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have.

Verification is Static Testing.

Activities involved in verification:

* Inspections
* Reviews
* Walkthroughs
* Desk-checking

**Metrics**



**Software Measurements**

Software Measurement is indicator of size, quantity amount or dimension of particular attribute of a product or process.

It helps the project manager & entire software team to take decisions that lead to successful completion of the project by generating quantity result.

* **Software measurements are of two categories:**

1. Direct Measures: It include software processes like Cost & Effort applied, Lines of code produced, Execution speed & Total no. of errors that have been reported.

2. Indirect Measures: It include products like Functionality. Quality, Complexity, Reliability. Maintainability and many more.

**Software Metrics**

Software Metrics provide measures, functions of formulas for various aspects of software process & product.

It including measuring software performance, planning work items, measuring productivity, and many other uses.

* Software metrics are of three categories:

1. **Product Metrics**: It estimate Size, Complexity, Quality & Reliability of software.

2. **Process Metrics**: It estimate Faults rate during development, Pattern of testing defect arrival, Time it takes for a fixed operation.

3. **Project Metrics**: It estimate Number of software developers, Cost, Scheduling and Productivity of software.

For Further details like LOC and FP please go through the link

https://www.youtube.com/watch?v=xLQL5q1JBTY