MODULE 3

DESIGN ENGINEERING

Module 3: Design Engineering

Unit No.	Topics	Hrs.	CO
3.1	Design Concepts, Design Principles	6	CO3
3.2	Architecture Design, Component Level Design, System Level Design, User Interface Design		

SOFTWARE DESIGN

- Once the requirement document regarding the software to be developed is presented, the phase of software design gets started.
- Software designing is considered as the initial phase of transforming the problem into a solution.
- In the design phase, all the relevant entities such as the customer, business requirements and technical considerations collaborate to formulate a product or a system.
- In design process, there are several elements such as set of principles, concepts and practices, which help a software engineer to model the system or product which to be built.
- The design model is assessed for quality and reviewed before generation of code and execution of tests.

SOFTWARE DESIGN

- The design model gives detailed information regarding software data structures, architecture, interfaces and components which are necessary to employ the system.
- IEEE defines software design as 'both a process of defining the architecture, components, interfaces, and other characteristics of a system and the result of that process.'
- In the design phase, important and strategic decisions are made to get the expected functionality and quality of the system.
- These decisions are considered to successfully develop the software and handle its maintenance in a way that the quality of the end product will be improved.

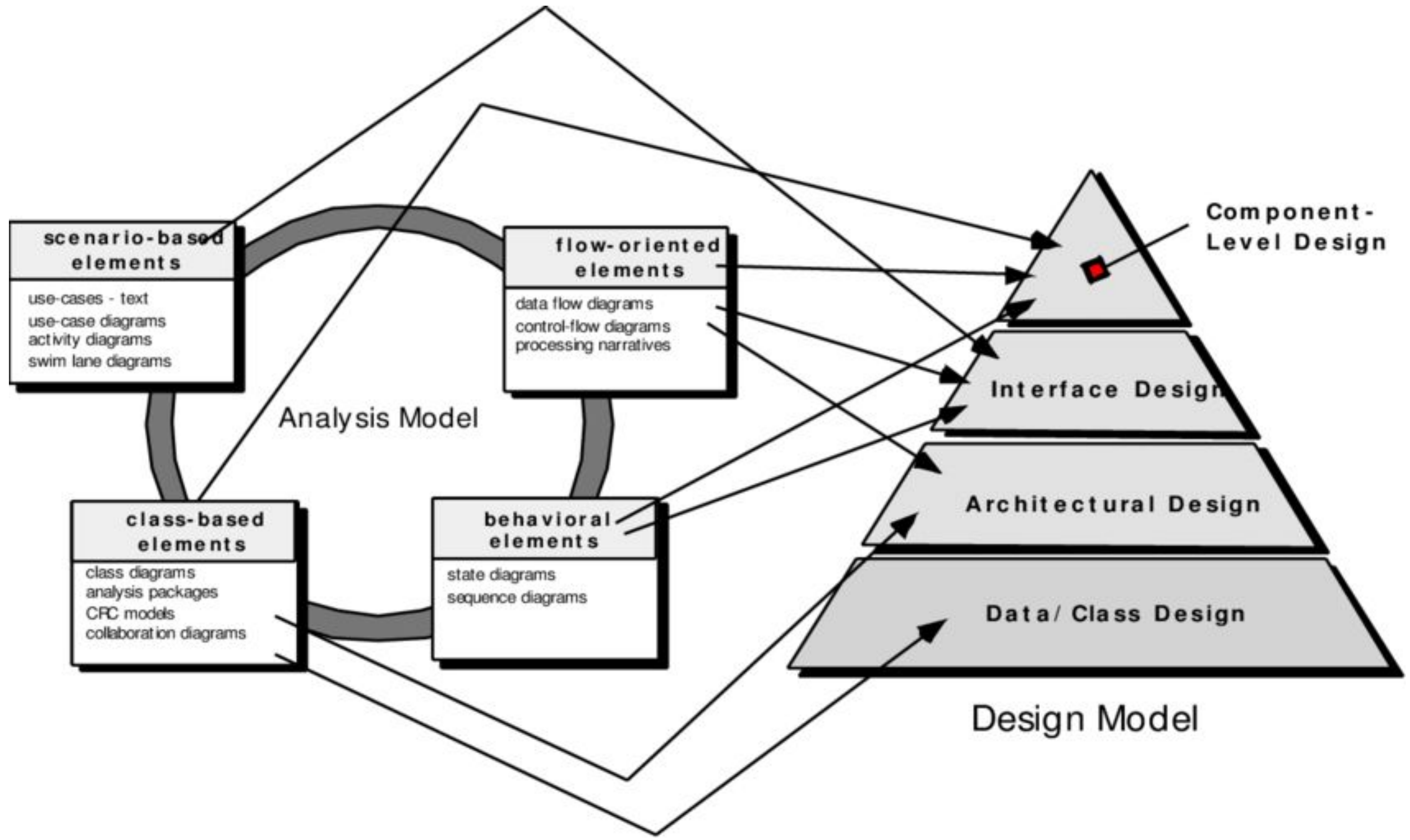
Good software Design should exhibit :

- Firmness: A program should not have any bugs that inhibit its function
- Commodity: A program should be suitable for the purpose for which it was intended .
- Delight : The experience of using the program should be pleasurable one .

DESIGN MODEL

Software design model consists of 4 designs :

- Data / class design
- Architectural design
- Interface design
- Component design


scenario-based elements
use-cases - text
use-case diagrams
activity diagrams
swim lane diagrams
flow-oriented elements
data flow diagrams
control-flow diagrams
processing narratives
class-based elements
class diagrams
analysis packages
CRC models
collaboration diagrams
behavioral elements
state diagrams
sequence diagrams
Analysis Model
Component-Level Design
Interface Design
Architectural Design
Data/ Class Design
Design Model

QUALITIES OF GOOD DESIGN

- Innovative
- Functional
- Honest
- User-oriented
- correctness

DESIGN PRINCIPLES

1. The design process should not suffer from “tunnel vision”.
2. The design should be traceable to the analysis model.
3. The design should not reinvent the wheel.
4. The design should “minimize the intellectual distance” between the software and the problem in the real world.
5. The design should exhibit uniformity and integration.
6. The design should be structured to accommodate change.
7. The design should be structured to degrade gently.
8. Design is not coding.
9. The design should be assessed for quality.
10. The design should reviewed to minimize conceptual errors.

DESIGN PRINCIPLES

1)The design process should not suffer from "tunnel vision." A good designer should consider alternative approaches, judging each based on the requirements of the problem, the resources available to do the job.

2) The design should be traceable to the analysis model. Because a single element of the design model often traces to multiple requirements, it is necessary to have a means for tracking how requirements have been satisfied by the design model.

3) The design should not reinvent the wheel. Systems are constructed using a set of design patterns, many of which have likely been encountered before. These patterns should always be chosen as an alternative to reinvention. Time is short and resources are limited! Design time should be invested in representing truly new ideas and integrating those patterns that already exist.

4) The design should "minimize the intellectual distance" between the software and the problem as it exists in the real world. That is, the structure of the software design should (whenever possible) mimic the structure of the problem domain

- 5)**The design should exhibit uniformity and integration.** A design is uniform if it appears that one person developed the entire thing. Rules of style and format should be defined for a design team before design work begins. A design is integrated if care is taken in defining interfaces between design components.
- 6)**The design should be structured to accommodate change.** The design concepts discussed in the next section enable a design to achieve this principle.
- 7)**The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered.** Well- designed software should never "bomb." It should be designed to accommodate unusual circumstances, and if it must terminate processing, do so in a graceful manner.
- 8) **Design is not coding, coding is not design.** Even when detailed procedural designs are created for program components, the level of abstraction of the design model is higher than source code. The only design decisions made at the coding level address the small implementation details that enable the procedural design to be coded.
- 9) **The design should be assessed for quality as it is being created, not after the fact.** A variety of design concepts and design measures are available to assist the designer in assessing quality.
- 10) **The design should be reviewed to minimize conceptual (semantic) errors.** There is sometimes a tendency to focus on minutiae when the design is reviewed, missing the forest for the trees. A design team should ensure that major conceptual elements of the design (omissions, ambiguity, inconsistency) have been addressed before worrying about the syntax of the design model.

QUALITY ATTRIBUTES

- The attribute of design name as 'FURPS' are as follows:

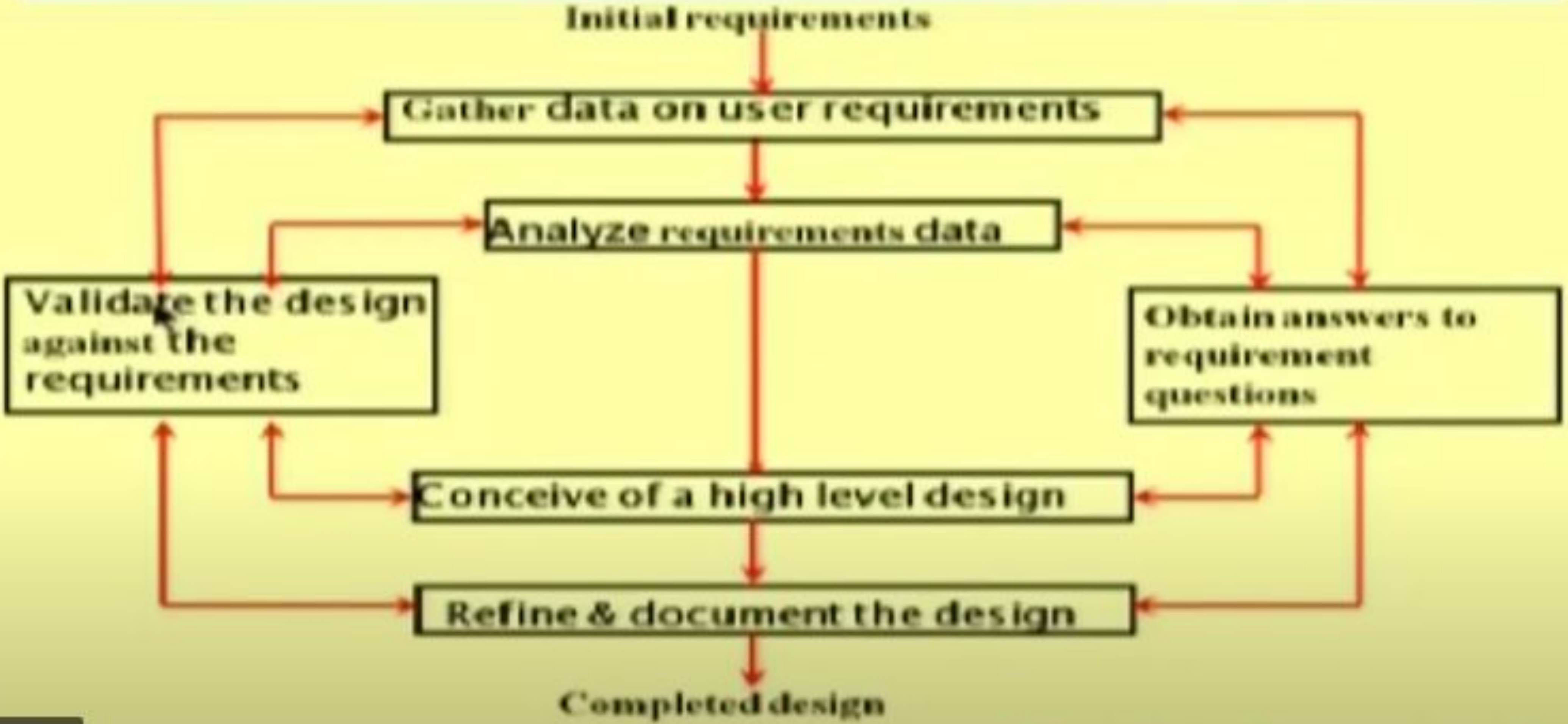
  Functionality:
  It evaluates the feature set and capabilities of the program.

  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 the program predictability. is measured by considering processing speed, response time, resource consumption, throughput and efficiency.

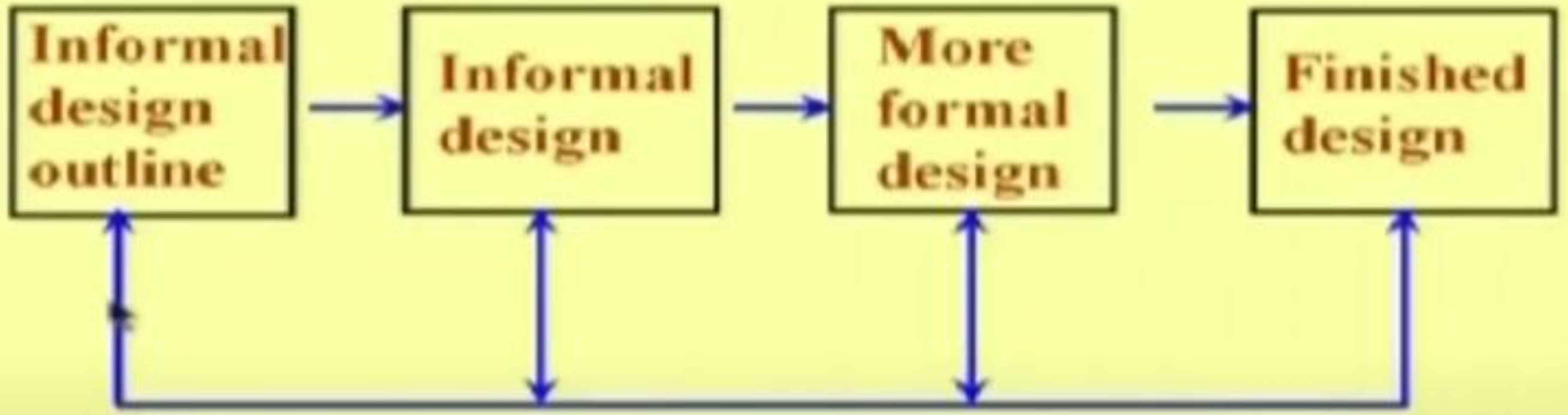
- Performance:
  It is measured by considering processing speed, response time, resource consumption, throughput and efficiency.
- Supportability:It combines the ability to extend the program, adaptability, serviceability. These three term defines the maintainability.
- Testability, compatibility and 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.

Software Design


Initial requirements
Gather data on user requirements
Analyze requirements data
Validate the design against the requirements
Obtain answers to requirement questions
Conceive of a high level design
Refine & document the design
Completed design

Design framework

Software Design



The transformation of an informal design to a detailed design.

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.
- 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.

Q. WHAT ARE DESIGN CONCEPTS ?

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

DESIGN CONCEPTS

- The set of fundamental software design concepts are as follows:

DESIGN CONCEPTS: 1.ABSTRACTION

- Abstraction is one of the fundamental ways that we as humans cope with complexity.
- IEEE defines abstraction as "a view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information."
- There are two ways to use the concept of abstraction; as a process and as an entity.
  - As a process, abstraction refers to a mechanism of hiding irrelevant extra details and representing just the important essential features of an element so that one can concentrate on essential things at a time.
  - As an entity, abstraction refers to a model or view of an item.

- In the software process, all the steps are accomplished via several levels of abstraction.
- At the highest level, one can get the outline of the solution to the respective problem. While at the lower levels, he/she will get the solution to the problem in detail.
- For example, in the requirements analysis phase, we can understand the outline of the problem and as one proceeds during the software process, the abstraction level reduces and source code of the software is generated at the lowest level
- An example of a procedural abstraction would be the word OPEN for a DOOR .
- Attributes : stand from place- go to door-move the knob-open door.
- Data abstraction is door.

DESIGN CONCEPTS: 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.
- Software architecture refers to the structure of the system, which consists of several components regarding a program system, the attributes (properties) of those components and the relationship among them.
- The software architecture helps the software engineers in the process of analyzing the software design.
- Additionally it also helps the software engineers in decision-making as well as handling risks.

- The Architectural design can be represented using number of models

- Structural model –represents architecture as an organize collection of program component .
- Framework model –Increases the level of abstraction by attempting to identify repeatable
- Dynamic model –addresses behavioral aspects of the program architecture ,indicating how the system configuration may change as a function of external event.
- Process model –focuses on business and technical process.

DESIGN CONCEPTS:3.PATTERNS

A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

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.

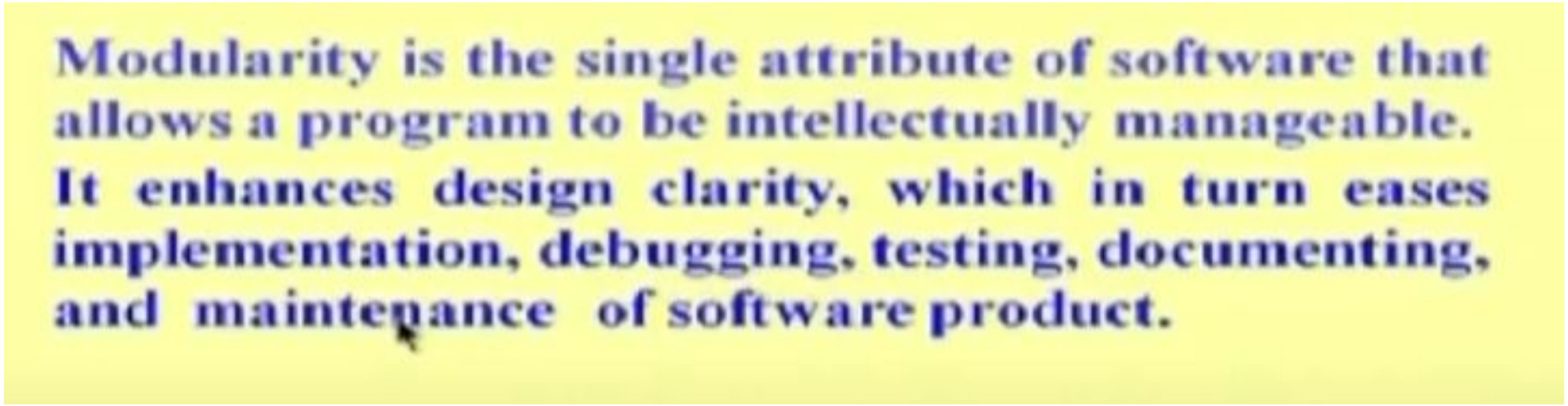
2.whether the pattern can be reused

3.whether the pattern can serve as guide for developing similar or different pattern.

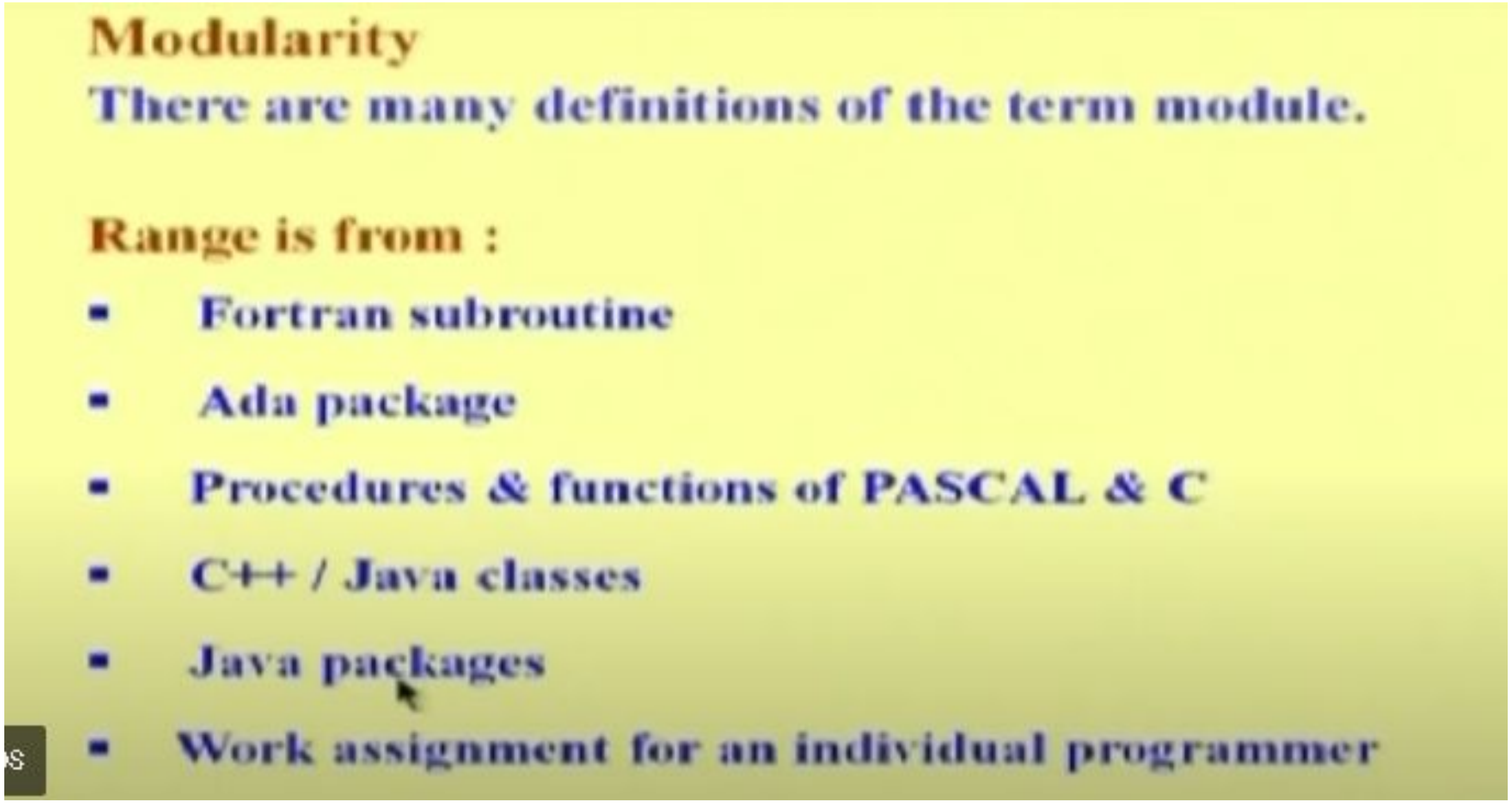
DESIGN CONCEPTS: 4.MODULARITY

- 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.
- Modularity can be achieved by the process of dividing the software into components which are uniquely named and addressable. These components are also known as modules.
- A complex system (huge application is partitioned into a set of discrete modules in such a manner that one should be able to develop those modules independent of each other. After development is completed, the modules are integrated to form an entire project.
- One has to remember that as the number of modules a system is divided into increases, it becomes easy to handle the software but it also increases the effort required to integrate the modules.

- The process of modularizing a design helps to plan the development in a more efficient way, accommodate changes without difficulty, conduct testing and debugging effectively as well as efficiently, and carry out the maintenance work without adversely affecting the functionality of the software.


Modularity is the single attribute of software that allows a program to be intellectually manageable.

It enhances design clarity, which in turn eases implementation, debugging, testing, documenting, and maintenance of software product.



Modularity

There are many definitions of the term module.

Range is from :

- Fortran subroutine
- Ada package
- Procedures & functions of PASCAL & C
- C++ / Java classes
- Java packages
- Work assignment for an individual programmer

DESIGN CONCEPTS: 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
- It is important to specify and design modules in such a manner that the data structures as well as processing details of one module should not be accessible to any other modules.
- Only required information is transferred between the modules. The way of hiding unnecessary details is referred to as information hiding.
- IEEE defines information hiding as "the technique of encapsulating software design decisions in modules in such a way that the module's interfaces reveal as little as possible about the modules inner workings; thus each module is a black box' to other modules in the system.
- Information hiding is an important aspect when there is need of modifications during the testing and maintenance phase

DESIGN CONCEPTS: 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 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.

COHESION

- The measure of how strongly the elements are related functionally inside a module is called cohesion
- In software engineering and the elements inside a module can be instructions, groups of instructions, definition of data, call from another module etc.
- The aim is always for functions that are strongly related and the expectation is for everything inside the module to be in connection with one another.
- Basically, cohesion is the internal glue that keeps the module together.
- Good system design must have high cohesion between the components of the system.

DIFFERENT TYPES OF COHESION

Co-incidental cohesion: (worst)

- An unplanned cohesion where elements are not related(unrelated).
- The elements have no conceptual relationship other than location in source code.
- It is accidental and the worst form of cohesion
- Eg.- print next line and reverse the characters of a string in a single component.

Logical cohesion:

- When logically classified elements are combined into a single module but not functionally then it is called as logical cohesion.
- Here all elements of modules contribute to same system operation.
- Eg.- Print Functions: the case where a set of print functions generating different output reports are arranged into a single module.

Temporal Cohesion:

- The elements are related by their timing involved.
- A module connected with temporal cohesion all the tasks must be executed in the same time-span.
- This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.
- Eg.- a function which is called after catching an exception which closes open files, creates an error log, and notifies the user.

Procedural Cohesion:

- A module is said to possess procedural cohesion, if the set of functions of the module are all part of a procedure (algorithm) in which certain sequence of steps have to be carried out for achieving an objective.
- Eg.- a function which checks file permissions then opens the file.

Sequential Cohesion:

- Sequential cohesion is when parts of a module are grouped because the output from one part is the input to another part like an assembly line (e.g. a function which reads data from a file and processes the data).
- Sequential cohesion is easy maintenance
- Eg.- In a transaction processing system (TPS), the get-input, validate-input, sort-input functions are grouped into one module.

Communication cohesion:

- When elements of module perform different functions but each function accepts same input and generates same output then that module is said to be communicational cohesive.
- Eg.- Module determines customer details like use customer account no to find and return customer name and loan balance.

Functional cohesion (best)

- If elements of module are grouped together since they contribute to a single function then with module is functionally cohesive module.
- All elements of such a module are necessary for successful execution of function.
- Eg.- Read transaction record or Assign seat to airline passenger

COUPLING

- Coupling is the measure of the degree of interdependence between the modules.
- A good software will have low coupling.
- The lower the coupling, the more modular a program is, which means that less code has to be changed when the program's functionality is altered later on.
- However, coupling cannot be completely eliminated; it can only be minimized.

TYPES OF COUPLING

Content Coupling (Worst):

- In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module.
- E.g.- a branch from one module into another module.

Common Coupling:

- The modules have shared data such as global data structures.
- The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change.
- Eg.- when two classes access the same shared data (e.g., a global variable).

Control coupling:

- When one function controls the flow of another function.

Data Coupling:

- If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled.
- In data coupling, the components are independent to each other and communicating through data.

Stamp Coupling :

- In stamp coupling, the complete data structure is passed from one module to another module.

External Coupling:

- In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware.

WHAT ARE THE BENEFITS OF HIGH COHESION AND LOW COUPLING?

- Benefits of high cohesion:
  - Readability: Closely related functions are contained in a single module.
  - Maintainability: Debugging tends to be contained in a single module
  - Reusability: Classes that have concentrated functionalities are not polluted with useless functions.
- Benefits of low coupling:
  - Readability: Classes that need to be analyzed are kept at a minimum.
  - Maintainability: Changes are confined in a single module.
  - Testability: Modules involved in unit testing are kept at a minimum

DIFFERENCE:

Coupling	Cohesion
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.
Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.
Coupling shows the relative **independence** between the modules.	Cohesion shows the module's relative **functional** strength.
While creating, you should aim for low coupling, i.e., dependency among modules should be less.	While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.
In coupling, modules are linked to the other modules.	In cohesion, the module focuses on a single thing.

DESIGN CONCEPTS: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.
- Refinement causes the designer to elaborate on the original statement ,providing more and more detail as each successive elaboration occurs .

DESIGN CONCEPTS: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.
- When software refactored ,the existing design is examined for redundancy ,unused design element, insufficient algorithm, inappropriate data structure etc.

DESIGN CONCEPTS: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.
- Five different types of design classes , each representing a different layer of the design architecture are suggested.

FIVE DIFFERENT TYPES OF DESIGN CLASSES

- 1. User interface classes
- These classes are designed for Human Computer Interaction(HCI).
- These interface classes define all abstraction which is required for Human Computer Interaction(HCI).
- 2. Business domain classes
- These classes are commonly refinements of the analysis classes.
- These classes are recognized as attributes and methods which are required to implement the elements of the business domain.
- 3. Process classes
- It implement the lower level business abstraction which is needed to completely manage the business domain class.
- 4. Persistence classes
- It shows data stores that will persist behind the execution of the software.
- 5. System Classes
- System classes implement software management and control functions that allow to operate and communicate in computing environment and outside world.