GANPAT UNIVERSITY

U. V. PATEL COLLEGE OF ENGINEERING

2CEIT502
SOFTWARE ENGINEERING

UNIT 7

SOFTWARE DESIGN

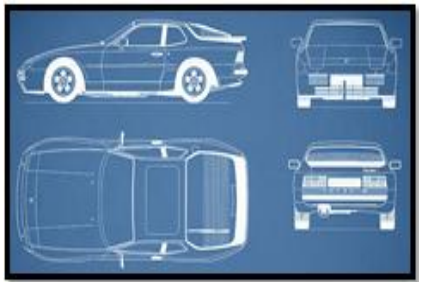
Prepared by: Prof. Y. J. Prajapati (Asst. Prof in IT Dept. , UVPCE)

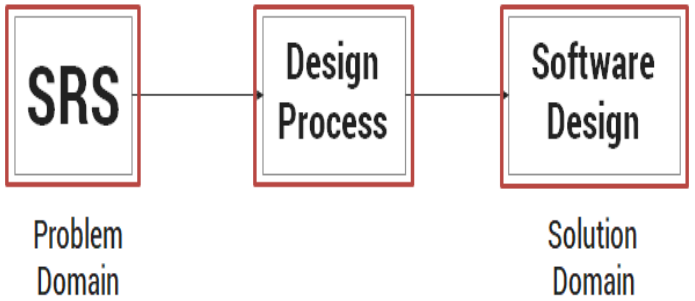
Outline

- What is Design ?
- Characteristics of good Design
- Design Model
- Conceptual Design & Technical Design
- The FURPS quality attributes
- Modularity
- Strategy of Design
- Function Oriented Design
- Object Oriented Design Approach (OOD)

What is Design ?

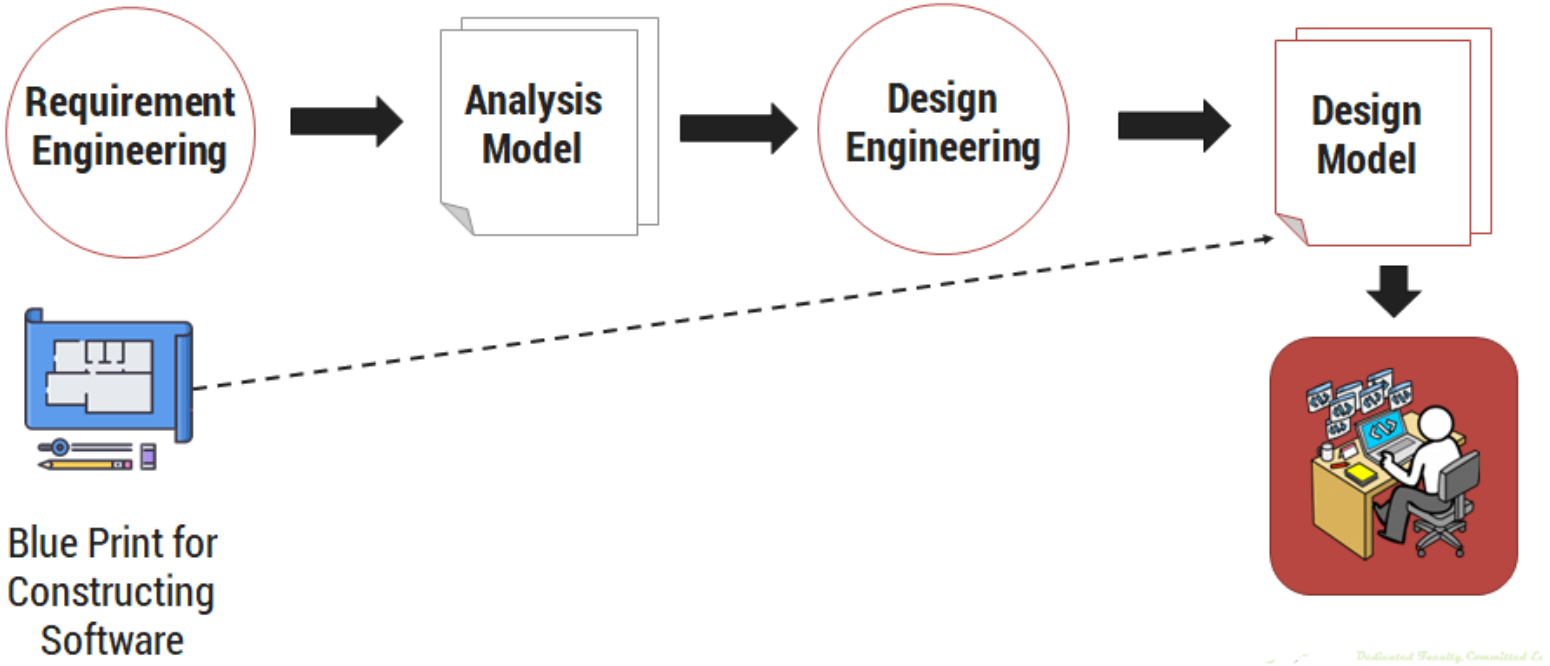
- A meaningful representation of something to be built
- It's a process by which requirements are translated into blueprint for constructing a software blueprint gives us the holistic view (entire view) of a software



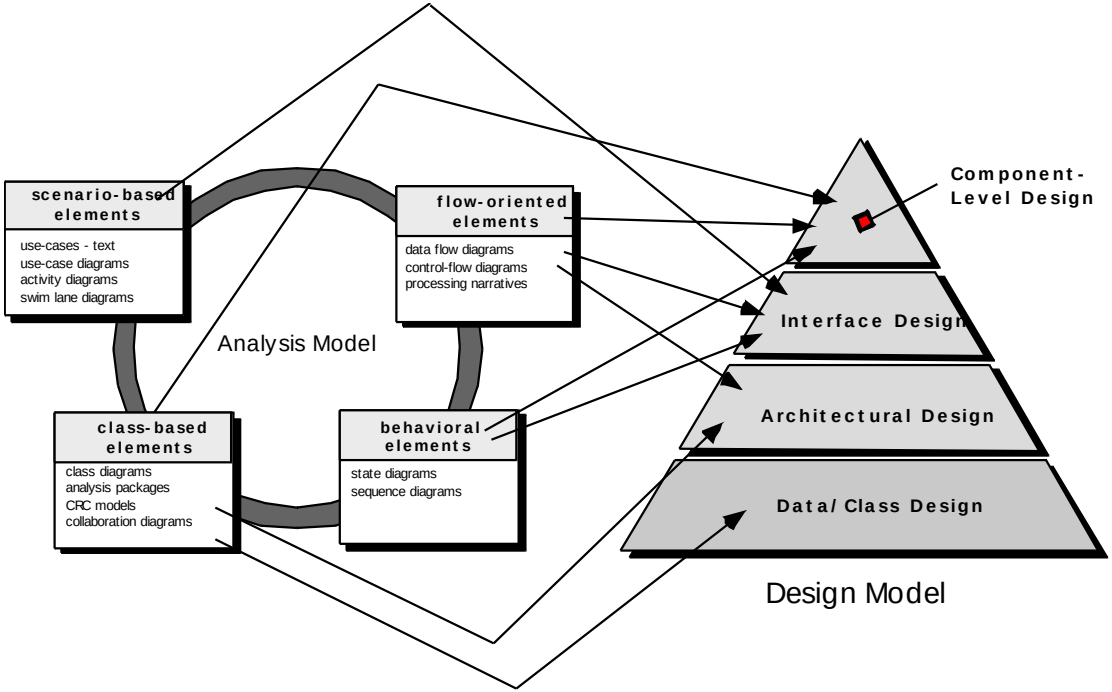


Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. It deals with representing the client's requirement, as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable using programming language.

Software Design Process ?



Analysis Model -> Design Model



Design Model

Data Design

It **transforms class models** into **design class** realization and **prepares data structure** (**data design**) required to implement the software.

Architectural Design

It **defines** the **relationship between** major **structural elements** of the software

Interface Design

It defines **how software communicates** with **systems** & with **humans**. An interface implies flow of information & behavior.

Procedural Design

It **transforms structural elements** of software into **procedural description** of software components

Characteristics of good Design

- The Design must be implement all explicit requirement available in requirements
- The Design must accommodate all implicit requirements given by stakeholders
- The design must be readable & understandable
- The good design should provide complete picture of software, addressing the data, functional and behavioral domains

Quality Guidelines

- A design should exhibit an architecture that (1) has been created using recognizable architectural styles or patterns, (2) is composed of components that exhibit good design characteristics and (3) can be implemented in an evolutionary fashion
- A design should be modular; that is, the software should be logically partitioned into elements or subsystems
- A design should contain distinct representations of data, architecture, interfaces, and components.
- A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
- A design should lead to components that exhibit independent functional characteristics.
- A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
- A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
- A design should be represented using a notation that effectively communicates its meaning.

Objective of Software design

Objectives of Software Design

Correctness: Software design should be correct as per requirement.

Completeness: The design should have all components like data structures, modules, and external interfaces, etc.

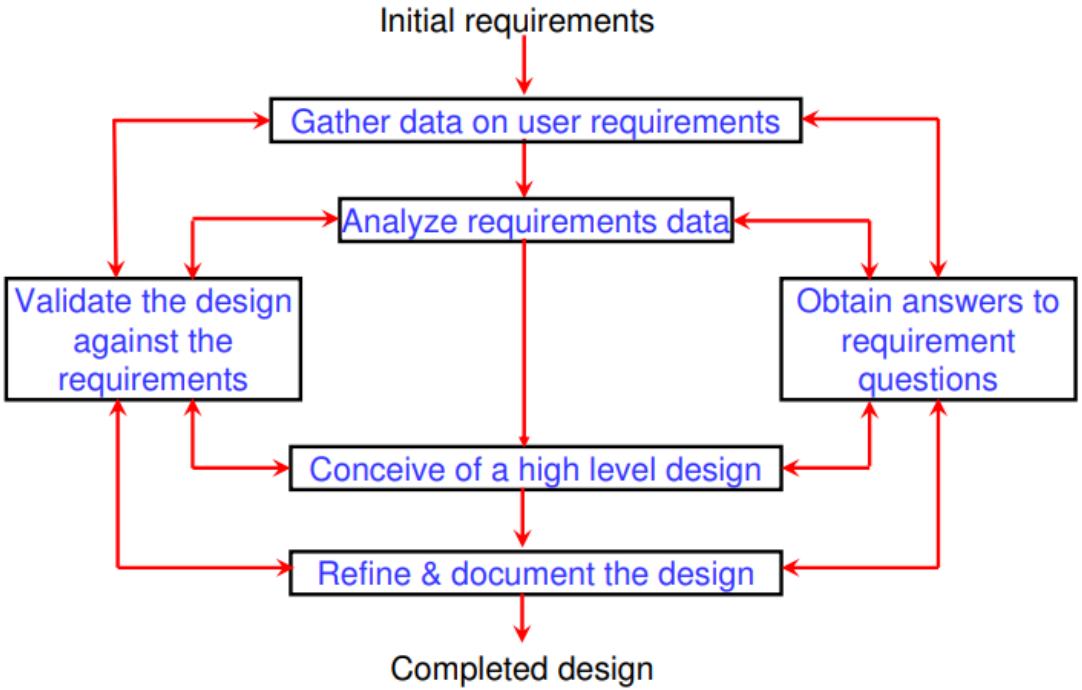
Efficiency: Resources should be used efficiently by the program.

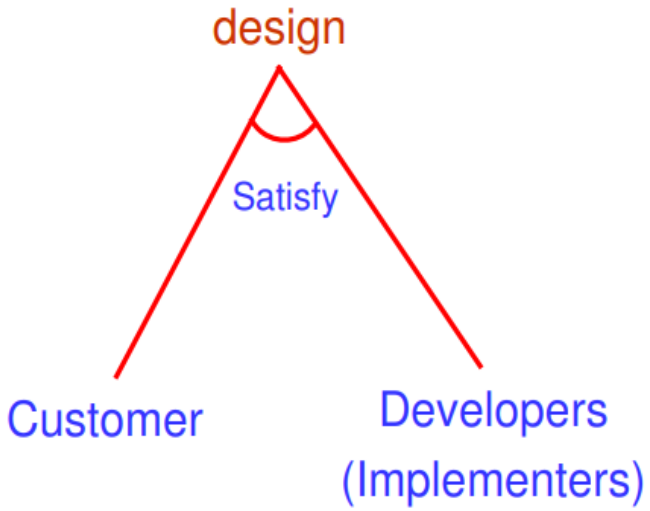
Flexibility: Able to modify on changing needs.

Consistency: There should not be any inconsistency in the design.

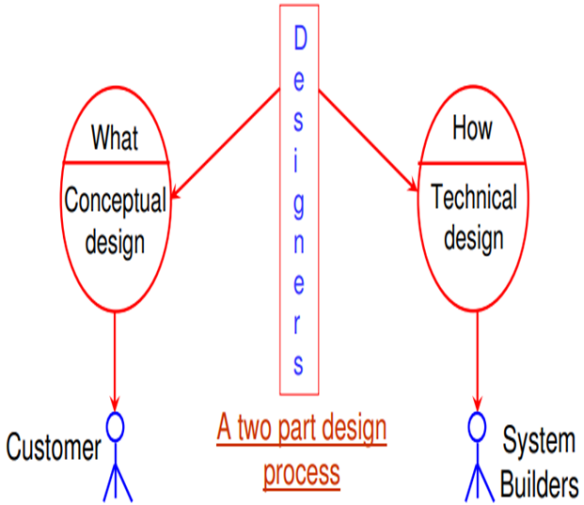
Maintainability: The design should be so simple so that it can be easily maintainable by other designers.

Design framework





Conceptual Design and Technical Design



Conceptual design answers :

- Where will the data come from ?
- What will happen to data in the system?
- How will the system look to users?
- What choices will be offered to users?
- What is the timings of events?
- How will the reports & screens look like?

Technical design describes :

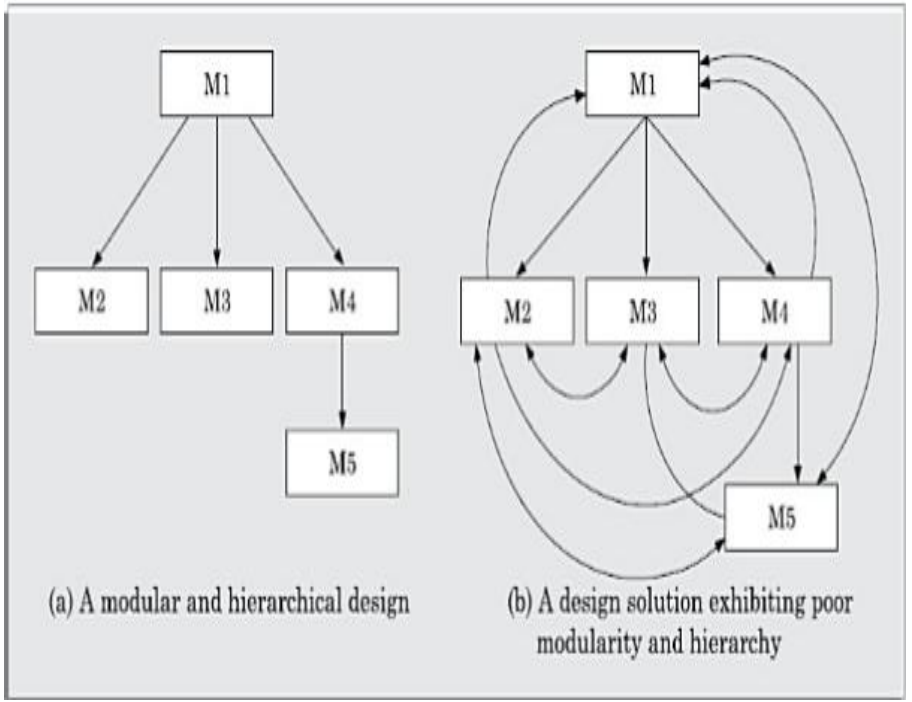
- Hardware configuration
- Software needs
- Communication interfaces
- I/O of the system
- Software architecture
- Network architecture
- Any other thing that translates the requirements in to a solution to the customer's problem.

Quality Attributes (FURPS)

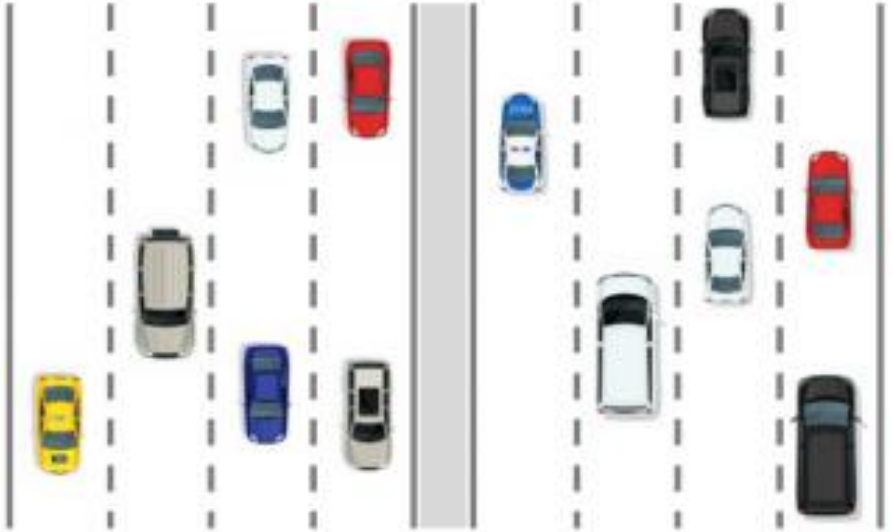
- Hewlett-Packard developed a set of software quality attributes that has been given the acronym FURPS: functionality, usability, reliability, performance, and supportability.
- ***Functionality*** is assessed by evaluating the **feature** set and **capabilities** of the program, the generality of the functions that are delivered, and the **security** of the overall system.
- ***Usability*** is assessed by considering human factors , overall **aesthetics**, **consistency**, and **documentation**.
- ***Reliability*** is evaluated by measuring the **frequency and severity of failure**, the accuracy of output results, the mean-time-to-failure (MTTF), the ability to **recover from failure**, and the **predictability** of the program.
- **Performance** is measured by considering processing speed, **response time**, resource consumption, **throughput**, and **efficiency.**
- **Supportability** combines the ability to extend the program (extensibility),adaptability, serviceability—these three attributes represent a more common term, maintainability—and in addition, testability, compatibility, configurability

Modularity

- **Modularity** specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable.
- The process of breaking down a software into multiple independent modules where each module is developed separately is called **Modularization**
- **The desirable properties of a modular system are:**
  - Each module is a well-defined system that can be used with other applications.
  - Each module has single specified objectives.
  - Modules can be separately compiled and saved in the library.
  - Modules should be easier to use than to build.



Example



Modularity

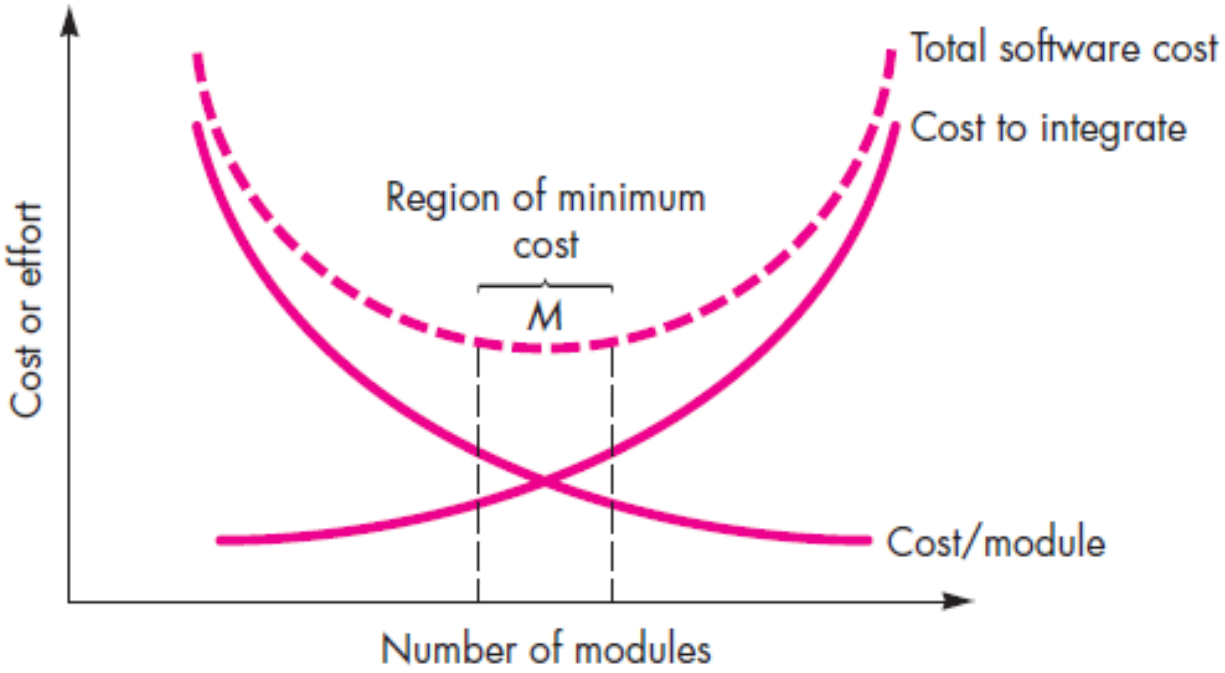
Advantages

- It allows large programs to be written by several or different people
- It encourages the creation of commonly used routines to be placed in the library and used by other programs.
- It simplifies the overlay procedure of loading a large program into main storage.
- It provides more checkpoints to measure progress.
- It provides a framework for complete testing, more accessible to test
- It produced the well designed and more readable program.

Disadvantages

- Execution time maybe, but not certainly, longer
- Storage size perhaps, but is not certainly, increased
- Compilation and loading time may be longer
- Inter-module communication problems may be increased
- More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done

Modularity and software cost



Coupling and Cohesion

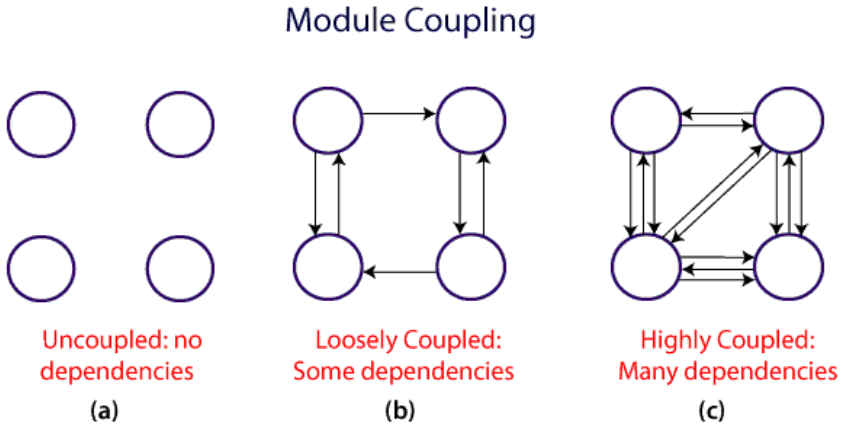
- When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together. There 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.

- **Module Coupling**
- In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other.
- **Module Cohesion**
- In computer programming, cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related.

Coupling and Cohesion

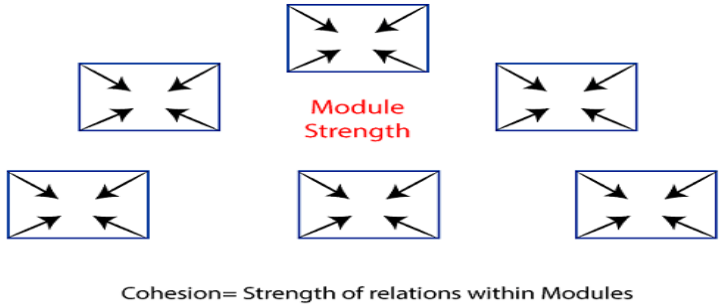
Module Coupling

- **Coupling:** Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.



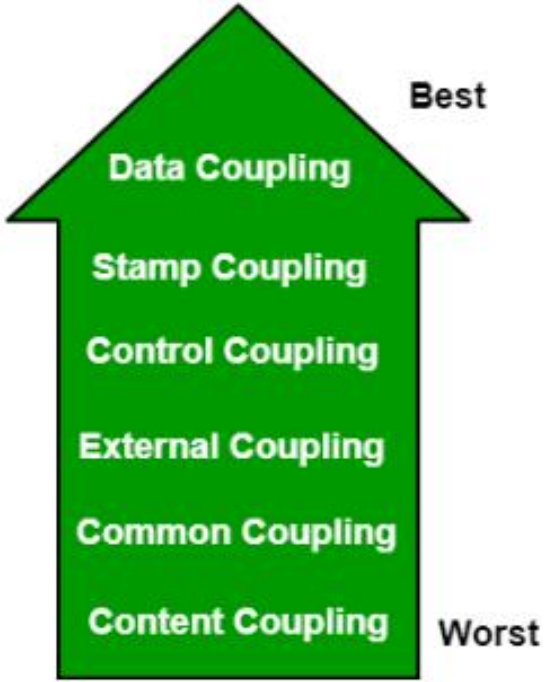
Module Cohesion

- Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



Types of Module Coupling

1. Data Coupling
2. Stamp Coupling
3. Control Coupling
4. External Coupling
5. Common Coupling
6. Content Coupling

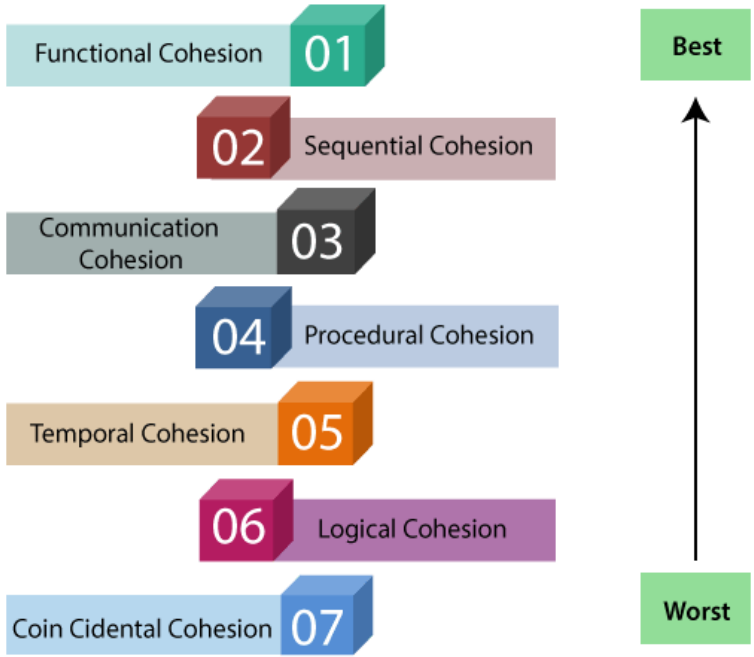


Types of Module Coupling

- **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. Module communications don't contain tramp data. Example-customer billing system.
- **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module.
- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behaviour and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
- **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. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduced maintainability.
- **Content Coupling:** 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. This is the worst form of coupling and should be avoided.

Types of Modules Cohesion

1. Functional cohesion
2. Sequential cohesion
3. Communicational cohesion
4. Procedural cohesion
5. Temporal cohesion
6. Logical cohesion
7. Co-incidental cohesion



Types of Modules Cohesion

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.
- **Sequential Cohesion:** An element outputs some data that becomes the input for other element
- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record into the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student CGPA, print student record, calculate CGPA, print CGPA.
- **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 initial time.
- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
- **Coincidental Cohesion:** The 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.

Differentiate between Coupling and Cohesion

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.

Strategy of Design

- A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.

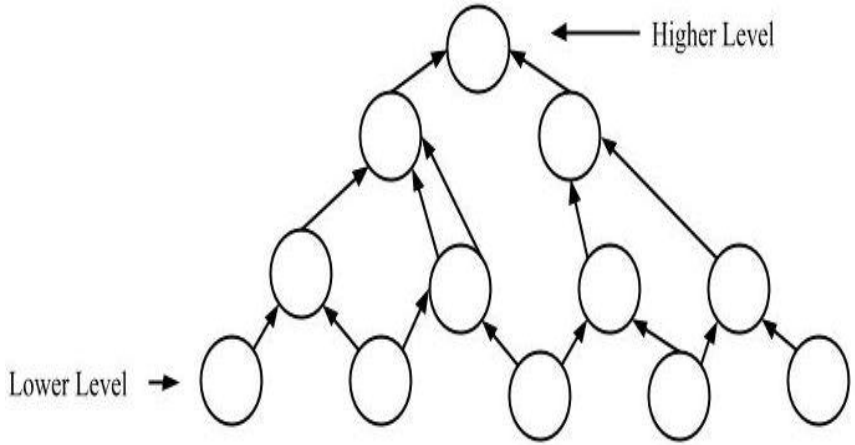
There are three possible approaches:

- Top-down Approach
- Bottom-up Approach
- Hybrid Design

Strategy of Design

- **Bottom-up approach**

  The design starts with the lowest level components and subsystems. By using these components, the next immediate higher level components and subsystems are created or composed. The process is continued till all the components and subsystems are composed into a single component, which is considered as the complete system

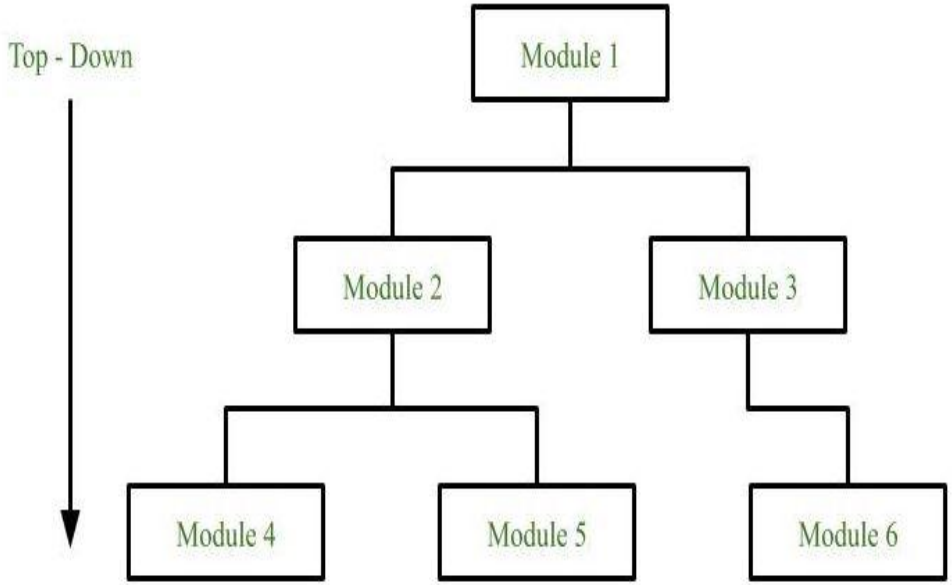


Strategy of Design

- **Top-down approach**

Each system is divided into several subsystems and components. Each of the subsystem is further divided into set of subsystems and components. This process of division facilitates in forming a system hierarchy structure. The complete software system is considered as a single entity and in relation to the characteristics, the system is split into sub-system and component. The same is done with each of the sub-system.

This process is continued until the lowest level of the system is reached. The design is started initially by defining the system as a whole and then keeps on adding definitions of the subsystems and components. When all the definitions are combined together, it turns out to be a complete system.



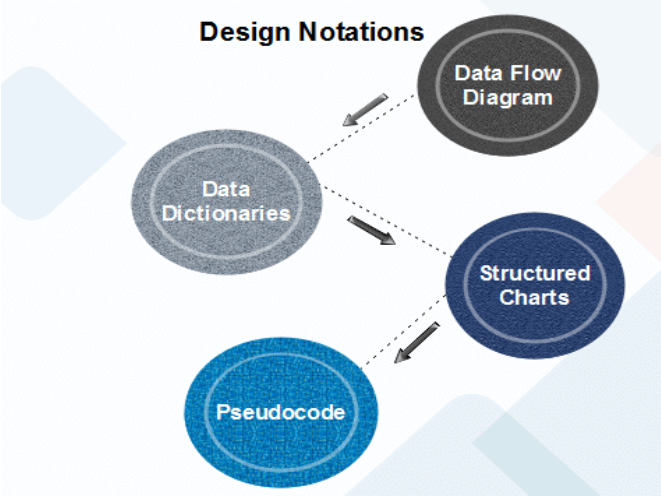
Strategy of Design

- **Hybrid Design**

  It is a combination of both the top – down and bottom – up design strategies. In this we can reuse the modules.

Function Oriented Design

- Function Oriented design is a method to software design where the model is decomposed into a set of interacting units or modules where each unit or module has a clearly defined function. Thus, the system is designed from a functional viewpoint.



- Data Flow Diagram
- Data-flow design is concerned with designing a series of functional transformations that convert system inputs into the required outputs. The design is described as data-flow diagrams. These diagrams show how data flows through a system and how the output is derived from the input through a series of functional transformations.
- A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

Function Oriented Design

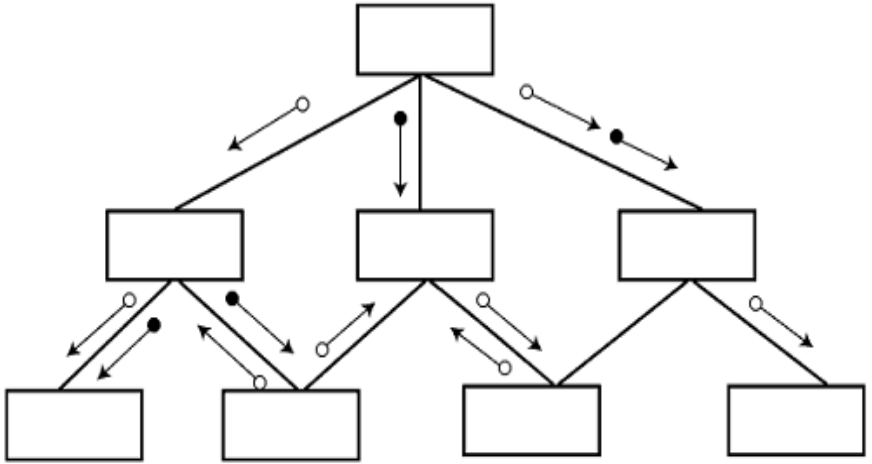
- Data Dictionaries
- Data dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirement stage, data dictionaries contains data items. Data dictionaries include Name of the item, Aliases (Other names for items), Description / purpose, Related data items, Range of values, Data structure definition / form.

- A data dictionary plays a significant role in any software development process because of the following reasons:
- A Data dictionary provides a standard language for all relevant information for use by engineers working in a project. A consistent vocabulary for data items is essential since, in large projects, different engineers of the project tend to use different terms to refer to the same data, which unnecessarily causes confusion.
- The data dictionary provides the analyst with a means to determine the definition of various data structures in terms of their component elements.

Function Oriented Design

- **Structured Charts**

- It is the hierarchical representation of system which partitions the system into black boxes (functionality is known to users but inner details are unknown). Components are read from top to bottom and left to right. When a module calls another, it views the called module as black box, passing required parameters and receiving results.
- Structured Chart is a graphical representation which shows:
  - System partitions into modules
  - Hierarchy of component modules
  - The relation between processing modules
  - Interaction between modules
  - Information passed between modules



Hierarchical format of a structure chart

Function Oriented Design

- **Pseudo-code**
- Pseudo-code notations can be used in both the preliminary and detailed design phases. Using pseudo-code, the designer describes system characteristics using short, concise, English Language phases that are structured by keywords such as If-Then-Else, While-Do, and End.
- It use keyword and indentation. Pseudo codes are used as replacement for flow charts

Object-Oriented Design

- An analysis model created using object oriented analysis is transformed by object oriented design into a design model that works as a plan for software creation. OOD results in a design having several different levels of modularity i.e., The major system components are partitioned into subsystems (a system level “modular”), and data their manipulation operations are encapsulated into objects (a modular form that is the building block of an OO system.).
- In addition, OOD must specify some data organization of attributes and a procedural description of each operation.


THANK YOU!
ANY QUESTIONS?