System Design

Syllabus

Unit	Contents	Lectures
6.	● **System Design** ● Introduction to system design ● The modeling of user requirement using System Decomposition and Modeling Work Flow Diagram Data Flow Diagram Entity Relationship Diagram System Flow Chart Converting E-R Diagram to Relational Tables	10

Introduction to System Design

- As we know that during the System Analysis, Requirement Definition and Description i.e. RDD and Software Requirement Specification i.e. SRS is taken into consideration and the Requirements of the User are Modelled using the following diagrams
  - Software Decomposing and Modelling
  - Work Flow Diagram
  - Data Flow Diagram
  - Entity Relation Diagram
  - System Flow Chart

Role of Software Decomposing and Modelling

- Provides a systems view in modular form using functional components.
- The system view can be of two types
  - Technical View as seen by the software engineer.
  - Business View as seen by the customer and the user.

Role of Work Flow Diagram

- Identify the interface relationship between different entities of the system.

Role of Data Flow Diagram

- The DFD drawn at different levels, show the flow of the data inputs through the system, laying down the path of converting basic input through various processes and applications to final functional goal achievement.

Role of Entity Relation Diagram

- Entity Relation Diagram show a quantitative dimension to the relation between two entities like
  - one to one
  - one to many
  - many to one
  - many to many

Role of System Flow Chart

- System Flow Chart shows how sub system work together to produce a system goal.

Introduction to System Design …

- The above five diagrams used to provide a precise understanding of the system and becomes the foundation for system design.
- The development of system design intends to build a design that can process micro data input to macro data output through different steps.
- An Engineering representation of customer and user requirement.
- It means the design can be traced back to the requirement and can be Assessed for quality specification of good design.

Introduction to System Design …

- During the design phase of Structured System Analysis and Design, following aspects are taken into consideration
    - Data and Data Structure
    - Data Model
    - Functional Modules
    - Components
    - Interfaces
    - System Components

Introduction to System Design …

- In the design phase, the focus is different in different stages.
- At the data and data structure level, the design focuses on entities, attributes and their relation and the nature of their use

Focus at the Data and Data Structure Level

- The design focuses on
  - Entities
  - Attributes
  - The Relation between Entities
  - The nature of their use

Focus at the Functional Level

- The design focuses on
  - Configuring the functions and features in the modules for an effective modular design of the system.

Focus at the Interface Level

- The design focuses on
  - Ergonomics that provides comfort to the users of the system.

Focus at the System Component Level

- The design focuses on
  - The evolution of an effective program structure through product design.

How to Design Data Dictionary?

- Data Dictionary is nothing but the data about the data.
- PAMDBS for the management of parking in sxc
  - Vehicle
  - Owner
  - User
  - Allocation
  - ParkingSpace
  - Parking

Software Design

- Design Engineering encompasses the set of principles, concepts and practices that lead to the development of high quality system or product.
- Design Principles guides the designer in the work that is performed.
- Design Concept has to be properly understood prior to the mechanics of the design practices is applied.
- Design Practice leads to the creation of representations of the software that serve as the guide for the construction activity that follows.

What is Design?

- Design is what virtually every engineer wants to do.
- It is the place where creativity rules, where customer requirements, business needs and rules and technical considerations come together to formulate a product or system.
- Design creates a representation or model of the software that provides detail about software data structure, architecture, interface and components that are required for the implementation of software.

What Role Does Design Play?

- Design allows a software engineer to model the system or product that is to be built.
- This model can be further assessed for quality improvement before the code generation.
- Tests are conducted and end users are involved.
- Design helps in the establishment of quality product.

What is done during Design?

- Architecture of the software or product must be implemented.
- Interface that connects the software to the end user, other systems, devices and own components have to be modeled.
- Software components that are used to construct the system are designed.

Work Product

- A design model that encompasses architectural, interface, component level and deployment representations is the primary **work product** that is produced during the software design.

Goal of Design

- To produce a model or representation that exhibits firmness, commodity and delight.
- To accomplish this, the designer must practice diversification and convergence.
- Both the diversification and convergence demand intuition and judgement, which are based on experience in building similar entities, principles and heuristics that guide the way in which the model evolves.

Design Model

- Data / Class Design
- Architectural Design
- Interface Design
- Component Level Design

Design Model: Data / Class Design

- The data or class design transforms analysis class models into design class realizations and the requisite data structures are required to implement the software.

Design Model: Architectural Design

- The architectural design defines the relationship between major structural elements of the software, the architectural styles and design pattern that can be used to achieve the requirements defined for the system and the constraints that affect the way in which architectural design can be implemented.
- The architectural design representation – framework a computer based system can be derived from the system specification, analysis model and the interaction of sub system defined within the analysis model.

Design Model: Interface Design

- The interface design describes how the software communicates with systems that incorporates with it and the user who uses it.
- An interface implies a flow of information i.e. data/control and specific type of behaviour.

Design Model: Component Level Design

- The component level design transforms structural elements of the software architecture into a procedural description of software components.
- The component design is done using
  - Class Based Models
  - Flow Models
  - Behavioral Model

Design Model: Component Level Design …

- There are two ways of software design
    - Simple Design
    - Complicated Design

Design Model: Component Level Design …

- During design, the main stress is on “**Quality**”.
- Design provides the representation of the software that can be assessed for quality.

Design Process

- Software design is an iterative process through which requirements are translated into “Blueprint” for the construction of the software.
- The blueprint represents the holistic view of the software i.e. the design is represented at a high level of abstraction – a level that can be directly traced to the specific system objectives and data, functional and behavioral requirements.
- With each iteration, subsequent refinements, design representations at lower levels of abstractions are achieved.

How to Evolve Good Design?

- The design must implement all the requirements contained in the analysis model and it must accommodate all of the implicit requirements desired by the customer.
- The design must be reliable, understandable guide for those who generate code and for those who test and subsequently support the software.
- The design should provide complete picture of the software, addressing the data, functional and behavioral domain from implementation perspective.

How to Evaluate Design?

- To evaluate the quality of a design representation, following technical criteria has to be established.
- A design must exhibit an architecture that
  - has been created using recognizable architectural styles/pattern.
  - is composed of components that exhibits good design characteristics.
  - Can be implemented in an evolutionary fashion, thereby facilitating implementation and testing.

How to Evaluate Design? …

- A design must be modular so that the software should be logically partitioned into elements or sub systems.
- A design must contain distinct representation of data, architecture, interface and component.
- A design must lead to data structure that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
- A design must lead to components that exhibit independent functional characteristics.

How to Evaluate Design? …

- A design must lead to interface that reduce the complexity of connections between components and with external environment.
- A design must be derived using an iterative method that is driven by information obtained during software requirement analysis.
- A design must be represented using a notation that effectively communicates it meaning.

Attributes of Good Design

- Hewlett – Packard developed a set of software quality attributes that has been given the acronym FURPS.
- The FURPS include
  - Functionality
  - Usability
  - Reliability
  - Performance
  - Supportability

Attributes of Good Design: Functionality

- It is assessed by evaluating the features set and capabilities of the program.
- The generality of the functions that are derived and the security of the overall system.

Attributes of Good Design: Usability

- It is assessed by considering human factors, overall aesthetics, consistency and documentation.

Attributes of Good Design: Reliability

- It 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.

Attributes of Good Design: Performance

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

Attributes of Good Design: Supportability

- It combines the ability to
  - Extend the Program (Extensibility)
  - Adaptability
  - Serviceability
- These three attributes represent more common term maintainability.

Design Concept

- A set of fundamental software design concepts has evolved over the history of software engineering.
- The degree of interest in each concept has varied over the years, each has stood the test of time.
- Each concept provides the software designer with a foundation from which more sophisticated design methods can be applied.
- “The beginning of wisdom for software engineer is to recognize the difference between getting a program to work and getting it right.”

Design Concept …

- Fundamental software design concepts provide the necessary framework of “**getting it right**”.
  - Abstraction
  - Architecture
  - Patterns
  - Modularity
  - Information Hiding (Encapsulation)
  - Functional Independence
  - Refinement
  - Refactoring

Design Concept …

- Design Classes
- Structure
- Verification
- Concurrency
- Aesthetics

Design within the Scope of Software Engineering

- Software design is the core of all software activities and is applied regardless of the software process model that is used. Once the software requirements have been analyzed and modeled, software design is the last software engineering activity within the modeling activity and sets the stage for the code generation and testing.
- According the Richard Due, “the most common miracle of software engineering is the translation of analysis to design and design to code”.
- Webster suggested that, the process of design includes conceiving and planning out in the mind and making a drawing pattern or sketch of.

Design within the Scope of Software Engineering …

- Software design has following components
  - External Design
  - Internal Design
    - Architectural Design
    - Detailed Design

External Design

- External design of software includes conceiving, planning out and specifying the externally observables characteristics of a software product.
- These characteristics include
  - User Interface
  - Report Format
  - External Data Sources
  - Data Sinks
  - Functional Characteristics
  - Performance Requirement

External Design …

    - High Level Process Structure
- External Design starts during the analysis phase and continue during design phase.
- External design is concerned with refining those requirements and defining the high level view of the system.
- Thus the distinction between requirement definition and external design is not sharp but is rather a gradual shift in emphasis from detailed “what” to high level “how”.

Internal Design

- The Internal Design involves conceiving, planning out and specifying the internal structure and processing details of the software product.
- The goals of internal design is to specify internal structures and processing details to record design decisions and indicate certain alternatives and thread offs were chosen to elaborate the test plan and provide a blue print for implementation, testing and maintenance activity.

Internal Design …

- Internal Design includes
    - Architectural Design
    - Detailed Design

Architectural Design

- Architectural Design concerns with designing the conceptual view of the system, identifying internal processing functions, decomposing high level functions into sub functions, designing internal data streams and data stores and establishing relationship and inter-connectivity among functions, data streams and data stores.

Detailed Design

- Detailed Design includes specifications of algorithms that implement the function, concrete data structure that implements the data store, the actual interconnection among functions and data structure and packaging for the system.

Detailed Design

- Detailed Design includes specifications of algorithms that implement the function, concrete data structure that implements the data store, the actual interconnection among functions and data structure and packaging for the system.

Design Concept …

- Fundamental software design concepts provide the necessary framework of “getting it right”.
  - Abstraction
  - Architecture
  - Patterns
  - Modularity
  - Information Hiding (Encapsulation)
  - Functional Independence
  - Refinement
  - Refactoring

Design Concept …

- Design Classes
- Structure
- Verification
- Concurrency
- Aesthetics

Abstraction

- Abstraction permits separation of conceptual aspects of the system from the implementation details.
- During software design, abstraction allows us to organize and channel our thought process by postponing structural considerations and detailed algorithmic considerations until the functional characteristic data streams and stores has been established.
- Structural considerations are then addressed prior to consideration of algorithmic details.

Abstraction …

- Architectural design specifications are models of software in which the functional and structural attributes of the system are emphasized.
- During detailed design, architectural structure is refined into implementation details.
- Design is thus a process of proceeding from abstract consideration to concrete implementation.

Abstraction …

- The abstraction mechanisms are
  - Functional abstraction
  - Data abstraction
  - Control abstraction

Functional Abstraction

- It includes the use of parameterized sub program.
- The ability to parameterize sub program and to bind different parameter values on different invocation of sub program is a powerful abstraction mechanism.
- It can be generalized through a collection of sub program called groups.
- Within a group, certain routines have the visibility property which allows them to be used with routine with the other group.

Functional Abstraction …

- Routines without visible properties are hidden from other group and can only be used within a contained group.
- A group thus provides a functional abstraction in which the visible routine communicates with the other group and the hidden routines exist to support the visible ones.

Data Abstraction

- It includes specification of data type or a data object by specifying legal operation on object; representation and manipulation details are expressed.
- The term "abstract datatype" is used to denote the declaration of the datatype from which numerous instances can be created.
- Abstract datatype are abstract in the sense that representation of the data items and implementation details of the function that manipulate the data items are hidden within the group that implements the abstract type.

Control Abstraction

- It is used to state desired effect without stating the exact mechanism of control.
- For example, if and while statement are the abstraction of machine code, implementation that involves, conditional jump instruction.
- At the architectural design level, control abstraction permits specification of sequential sub program, exception handling and co routine and concurrent program unit without concern of details of implementation.

Architecture

- Software architecture includes the overall structure of the software and the ways in which that structure provides conceptual integrity of the system.
- In its simplest form, architecture is the structure or organization of program components i.e. modules , the manner in which these components interacts and the structure of data that are used by the components.
- In broader sense, components can be generalized to represent major system elements and their interactions.

Architecture …

- The architectural design can be represented using models like
  - Structural Model
  - Framework Model
  - Dynamic Model
  - Process Model
  - Functional Model

Structural Model

- It represents architecture as an organized collection of program components.

Framework Model

- Framework is referred as implementation specific skeletal infrastructure.
- A framework is not an architectural pattern.
- It is rather a skeleton with a collection of “Plug Points”.
- The “Plug Points” enable the designer to integrate problem specific classes or functionality within the skeleton.
- Framework can also be referred as “Co-operating Classes”.

Dynamic Model

- It addresses the behavioral aspect of the program architecture, indicating how the structure or system configuration change as a function of external events.

Process Model

- It focus on the design of the business or technical process that the system must accommodate.

Functional Model

- It can be used to represent the functional hierarchy of the system.

Pattern

- Brad Appleton defines a design pattern, “A pattern is a named nugget of insight which conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns.”
- A design pattern describes a design structure that solves a particular design problem within a specific context and amid “forces” that may have an impact the manner in which the pattern is applied and used.

Pattern …

- The intent of the design pattern is to provide a description that enables a designer to determine
  - whether pattern is applicable to the current work.
  - whether the pattern can be reused.
  - whether the pattern can serve as guide for developing similar but functionally or structurally different pattern.

Encapsulation

- When a software system is designed using the encapsulation approach, each module in the system hides the internal details of its processing activities and module communicate only through well defined interfaces.
- In other words, modules should be specified and designed so that information i.e. data and algorithm contained within a module is inaccessible to other modules that have no need for such information.

Encapsulation …

- Hiding implies the effective modularity can be achieved by defining a set of independent modules that communicate with one another only that information necessary to achieve software function.
- Abstraction helps to define the procedural entities that make up the software where as hiding defines and enforces access constraints to both procedural detail within a module and any local data structure.

Concurrency

- The software system can be categorized as sequential or concurrent.
- In sequential system, only one portion of the system is active at any time.
- Concurrent systems have independent process that can be active simultaneously if multiple processes are available.
- In a single process, concurrent process can be interleaved in execution.

Concurrency …

- This permits implementation of time share, multi program and real time system.
- Problem unique to concurrent system includes
  - Deadlock
  - Mutual Exclusion
  - Synchronization

Deadlock

- When all the processes in the computing system is waiting for another processes to complete some action so that each can processed.

Mutual Exclusion

- It is necessary to ensure that the multiple processes do not attempt to update same components of shared processing state at the same time.

Synchronization

- It is necessary to ensure that the multiple processes do not attempt to update same components of shared processing state at the same time.

Verification

- It is required so that concurrent process operating at different execution speed can communicate with each other.
- Concurrency is the fundamental principle of software design because parallelism in software introduces added complexity and an additional degree of freedom into a design processing.

Verification

- Design is the bridge between customer requirement and implementation that satisfy those requirements.
- Verification refers to the set of activities that ensures that the software correctly implements a specific function.
- A design is said to be verified if it can be demonstrated that the design which result in implementation that satisfy the customer requirements.

Verification …

- This is done in two ways
  - Verification that the software requirement definition satisfies the customer’s need.
  - Verification that the design satisfies the requirement definition.

Aesthetics

- Aesthetic considerations are fundamental to design whether in art or technology.
- Simplicity, elegance, and clarity of purpose distinguish product of outstanding quality from mediocre product.
- When we speak of mathematical elegance or structural beauty, we are thinking of those properties that go beyond the satisfaction of requirement.
- It is difficult to list objective criteria for evaluating the aesthetic factors in a software product.

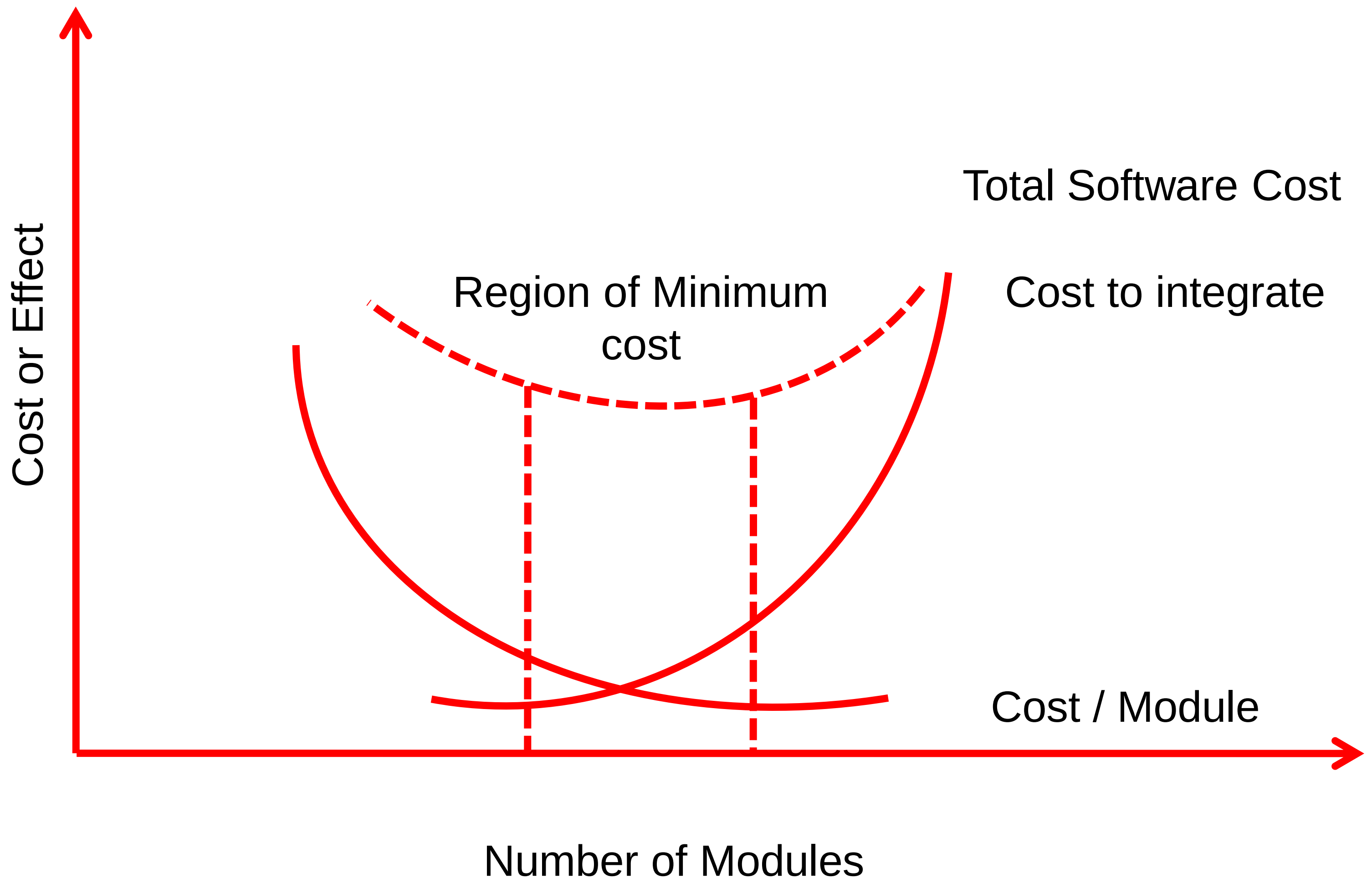
Modularity

- Modularity comes from the term "module". It is the work assigned to an individual. Modular system incorporates collection of abstraction in which each functional, data and control abstraction handles a local aspect of problem being solved.
- Properties of the modular structure
- Each processing abstraction is a well defined sub system i.e. potentially useful in other application.
- Each function in each abstraction has a single well defined purpose.

Modularity …

- Each function manipulates no more than one major data structure.
- Function shares global data selectively i.e. easy to identify all routines that shares a major data structure.
- Functions that manipulates instances of abstract data type are encapsulated with the data structure being manipulated.
- It is costly affair after a certain span of time.

Modularity …


Total Software Cost
Region of Minimum cost
Cost to integrate
Cost or Effect
Cost / Module
Number of Modules

Functional Independence

- The concept of functional dependence is a direct outgrowth of modularity and the concept of abstraction and information hiding.
- Functional Independence is achieved by developing modules with “single minded” functions and an “aversion” to excessive interaction with other modules.
- In other words, we want to design software so that each module addresses a specific sub function or requirements and has a simple interface when viewed from other parts of the program structure.

Functional Independence …

- It is fair to ask why independence is important.
- Software with effective modularity i.e. independent modules is easier to develop because function may be compartmentalized and interfaces are simplified.
- Independent modules are easier to maintain and test because secondary effects caused by design or code modification are limited, error propagation is reduced and reusable modules are possible.

Functional Independence …

- Hence functional independence is a key to good design and design is key to software quality.
- Independence is assessed during qualitative criteria
  - Cohesion
  - Coupling.

Functional Dependence

- The concept of functional dependence is a direct outgrowth of modularity and the concept of abstraction and information hiding.
- Functional Independence is achieved by developing modules with “single minded” functions and an “aversion” to excessive interaction with other modules. In other words, we want to design software so that each module addresses a specific sub function or requirements and has a simple interface when viewed from other parts of the program structure. It is fair to ask why independence is important.

Functional Dependence …

- Software with effective modularity i.e. independent modules is easier to develop because function may be compartmentalized and interfaces are simplified. Independent modules are easier to maintain and test because secondary effects caused by design or code modification are limited, error propagation is reduced and reusable modules are possible. Hence functional independence is a key to good design and design is key to software quality.

Functional Dependence …

- Independence is assessed during qualitative criteria
  - Cohesion
  - Coupling

Cohesion

- Cohesion is an indication of the relative functional strength of the module. It is a natural extension of information hiding concept.
- A cohesive module performs single task, requiring little interaction with other components in other parts of the program. It means a cohesive module should ideally do just one thing.

Types of Cohesion

- Co-Incidental Cohesion
- Logical Cohesion
- Temporal Cohesion
- Communicational Cohesion
- Sequential Cohesion
- Functional Cohesion
- Informal Cohesion

Co – Incidental Cohesion

- It occurs when the elements within a module have no apparent relationship to one another. This results in when a large monolithic program is modularized by arbitrarily segmenting the program into several small segments or modules or when a module is created from a group of unrelated instructions that appears several times in other modules.

Logical Cohesion

- It implies sub relationship among the elements in a module.
- Ex. A module that can perform all I/O operations or a module that edit data. A logically bound module often combines several related functions in a complex or an unrelated fashion.

Temporal Cohesion

- It exhibits the same disadvantages as logically bound module. However they are higher on the scale of binding because all elements are executed at same time and no parameters are required to determine which elements to execute. A typical example of temporal cohesion is a module that comprises the program initialization.

Communicational Cohesion

- The elements of the module processing communicational cohesion refers to the same set of I/O data. It is higher on the binding scale than the temporal binding because the elements are executed at one time and also refer to the same data.

Sequential Cohesion

- It occurs when the output of one element is the input of the next element and so on. For example read, next, update master file is sequentially bound. It is higher on the binding scale order because the module structure usually bears a class resemblance to the problem structure. Although a sequentially bound module can contain several function or a part of a function as the procedural processes in a program may be distinct from the function of a program.

Functional Cohesion

- It is strong and hence desirable. This type of binding of elements in a module because all kinds of elements in a module are related to the performance of the single function.

Informational Cohesion

- It occurs when the modules contain a complex Data Structure and several routines to manipulate the Data Structure. Each routine in a module exhibit functional binding also. It is also similar to functional cohesion and references to single data entity. Although it differs in the sense that communicational cohesion implies that all codes in the module executed on each invocation of the module. It requires only one functionally cohesive segment of the module to be executed on each invocation of the module.

High Cohesion

- A cohesive design class has a small, focused set of responsibilities and single mindedly applies attributes and methods to implement those responsibilities.

Coupling

- Coupling is an indication of the relative interdependence among modules. It is an indication of interconnection among modules in a software structure. Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module and what data pass across the interface. In software design, the stress is on “lowest possible coupling”. Simple connectivity among modules results in software that is easier to understand and less prone to a “ripple effect” caused when errors occur at one location and propagate throughout the system.

Types of Coupling

- Content Coupling
- Common Coupling
- Control Coupling
- Stamp Coupling
- Data Coupling

Content Coupling

- It occurs when one module modifies local data values, all instruction to other module. It can occur in assembly language programs.
- add(a,b);
- sub(a,b);

Control Coupling

- It involves the passing control flags between modules so that one module controls the sequence of processing in another module.
- sub(x,y)

```
  {
    if(x>y)
      return x – y;
    else
      return y – x;
   }
```

Stamp Coupling

- It is very much similar to common coupling. The difference between them is that global data items are shared among routines that requires the data. For example operator overloading.
- int add(int, int);
- void add();

Data Coupling

- It involves the passing of parameters to pass the data items between routines.
- printf(“%d + %d = %d”, a,b,add(a,b));

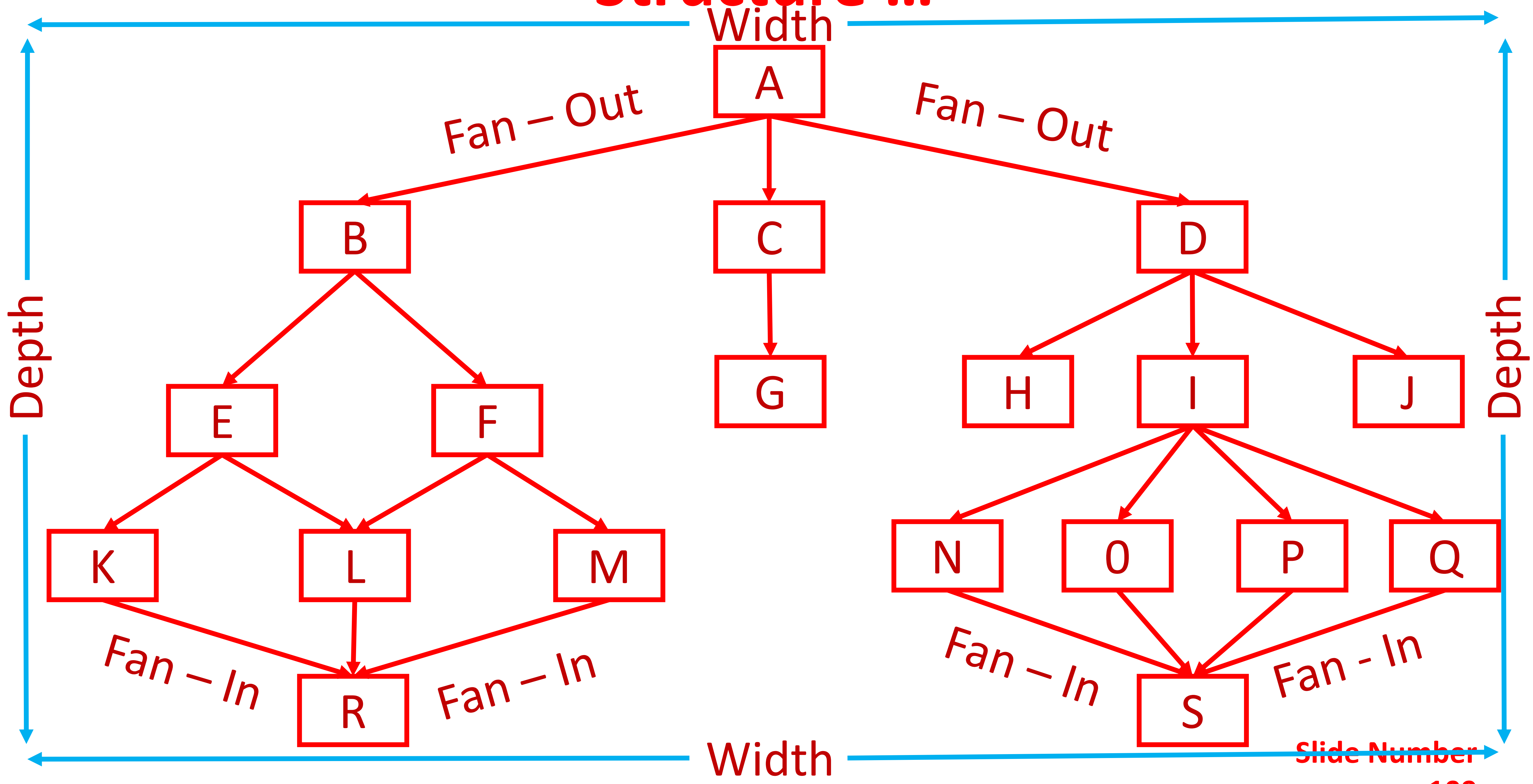
Low Coupling

- Within the design model, it is necessary for design classes to collaborate with one another. Although this collaboration has to be kept to an acceptable minimum level. If the design model is highly coupled i.e. all design classes collaborate with all other design classes, the system is difficult to implement, to test and to maintain over long period of time. Hence, design classes within the subsystem should have only limited knowledge of the classes in other subsystem. This is called “Law of Demeter” which suggests that a method should only send messages to those in the neighboring classes.

Structure

- It represents the organization of program components and implies a hierarchy of control. It does not represent procedural aspect of software like sequences of processes, order of decisions, and repetition of operations. We usually represent control hierarchy as
  - Depth
  - Width
  - Fan – In
  - Fan – Out

Structure …


Width
A
Fan – Out
Fan – Out
B
C
D
E
F
G
H
I
J
K
L
M
N
0
P
Q
R
S
Fan – In
Fan – In
Fan – In
Fan - In
Depth
Depth
Width

Structure: Depth

- It gives the number of level of controls.

Structure: Width

- It gives the overall span of control.

Structure: Fan – In

- It gives how many modules controlled a given module.

Structure: Fan – Out

- It gives number of modules controlled by a given module.

Visibility

- It indicates set of program components that may be involved by all used as data by a given component.

Connectivity

- It indicates set of components that are directly involved or used as data by a given component.

Structural Partitioning

- A program should be partitioned both horizontally and vertically.
  - Horizontal Partitioning
  - Vertical Partitioning

Horizontal Partitioning

- It defines separate branch of modular hierarchy for each major component.
- Simplest approach of horizontal partitioning is to define
  - Input Transformation
  - Data
  - Output
- As functions decouple one other, change tend to become less complex and extension to system less easily.
- It causes more data to pass across modules and complicates controls of program flow.

Vertical Partitioning

- It suggests that control and work should be distributed top down in program architectures.
- Top level module performs control functions and too little processing work and down level module performs input computation and output task.
- A change in control module cause more side effect to subordinate module then change in the control module has to be withdrawn.
- Due to this vertical partitioning, software architectures, they are less susceptible in side effect of changes made and so are more maintainable.

Data Structure

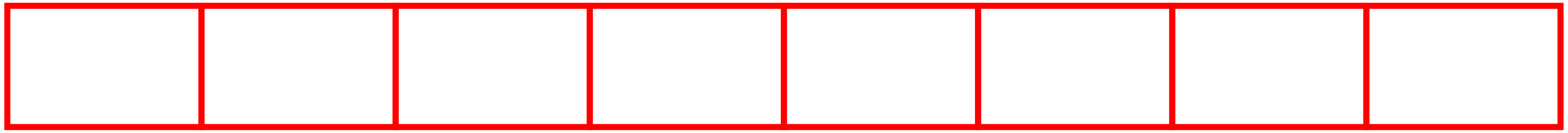
- It is the representation of logical relationship among individual elements of data. Data structure dictates organization, method of access, degree of associativity, processing, alternatives for information etc. there are number of classic data structure that act as building blocks. They are
  - Scalar
  - Sequential Vector
  - N – Dimensional Space
  - Linked List
  - Tree

Data Structure : Scalar Item

- It represents single element of information that may be addressed by an identifier or by specifying an address.

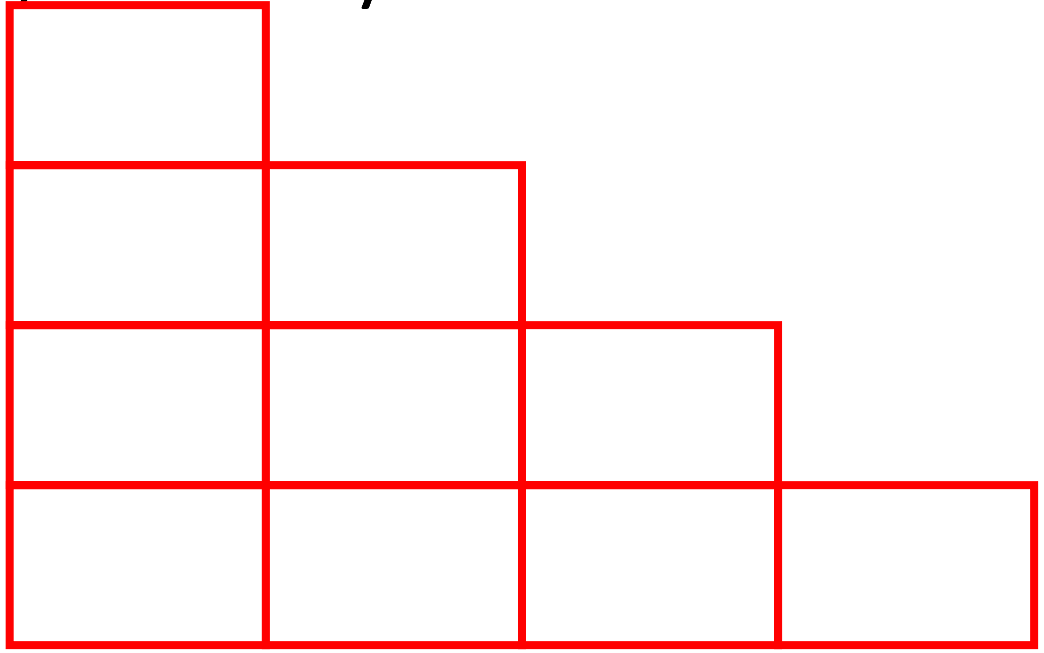
Data Structure : Sequential Vector

- It is formed when scalar items are organized in group.
- They are most common and open the concept of variable indexing of information.



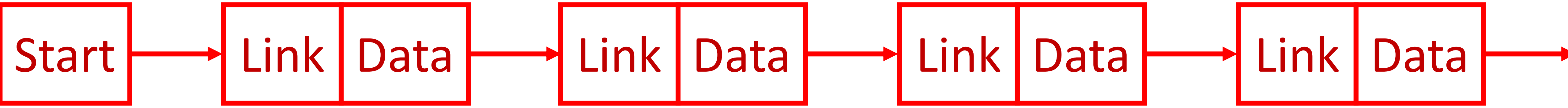
Data Structure : N – Dimensional Space

- It is created when a sequential vector is extended to two or more arbitrary ‘n’ numbers of dimensions.
- Most common is a 2D matrix. Some of the examples are Linked List, Hierarchy Tree etc.



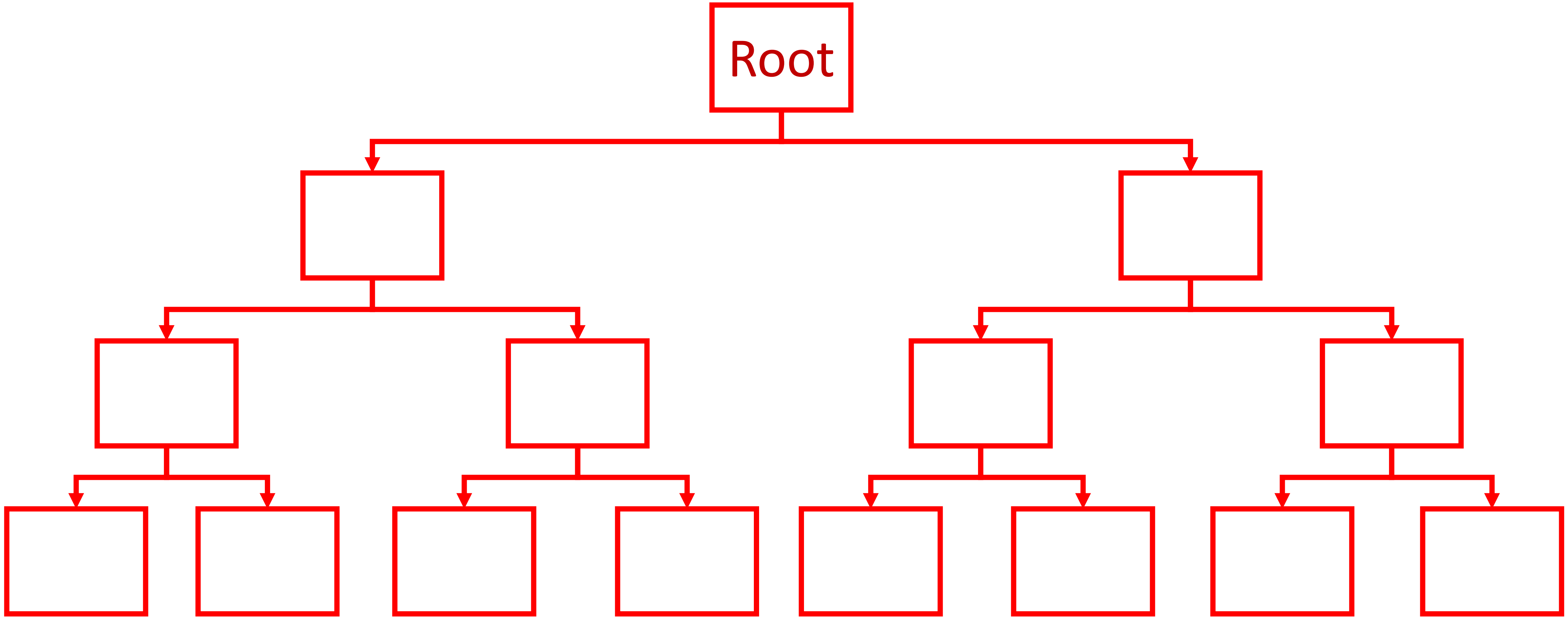
Data Structure : Linked List

- Linked List



Data Structure : Tree

- Tree



Refactoring

- Refactoring is the reorganization technique that simplifies the design or code of a component without changing its function or behaviour. Fowler defined refactoring as, “Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the design or code yet improves its internal structure.”
- When a software is refactored, the existing design is examined for redundancy, unused designed elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures or any other design limitations that can be corrected to produce a better design.

Design Class

- As we have seen that the analysis model defines a complete set of analysis classes. Each of these classes describes some element of the problem domain, focusing on the aspects of the problem that are user or customer visible. The level of abstraction of an analysis class is relatively high.
- As the design model evolves, the software team must define a set of design classes that
  - Refine the analysis classes by providing design details that will enable the classes to be implemented.

Design Class

- Create a new set of design classes that implement a software infrastructure to support the business solution.

- There are five different types of design classes, each representing a different layer of the design architecture. They are
  - User Interface Classes
  - Business Domain Classes
  - Process Classes
  - Persistent Classes
  - System Classes

Design Class : User Interface Class

- It defines all the abstractions that are necessary for human-machine interaction.
- In general, it occurs within the context of a metaphor i.e. check box, order form etc. and the design classes for the interface ,may be visual representation of the elements of the metaphor.

Design Class : Business Domain Class

- These are often refinements of the analysis classes defined earlier.
- The classes identify the attributes and services or methods that are required to implement some element of the business domain.

Design Class : Process Class

- Process Classes implement lower level business abstractions required to fully manage the business domain classes.

Design Class : Persistent Class

- Persistent Classes represent the data stores i.e. database that will persist beyond the execution of the software.

Design Class : System Class

- System Classes implement software management and control functions that enable the system to operate and communicate within its computing environment and with the outside world.

Design Class …

- As the design models evolve, the software team must develop a complete set of attributes and operations for each design class. The level of abstraction is reduced as each analysis class is transformed into a design representation. It means, analysis classes represent both objects and associated functions or methods that are applied to them using the requirement of the business domain.

Design Class …

- Design classes present a significantly more technical details as a guide for implementation. Each design class is reviewed to ensure that it is “well formed”.
- The characteristics that determine the classes to be “well formed or not” ,are
  - Complete and Sufficient
  - Primitiveness
  - High Cohesion
  - Low Coupling

Design Characteristics : Complete and Sufficient

- A design class should be complete encapsulation of all attributes and methods that can reasonably be expected to exist for the class.
- Sufficiency ensures that the design class contains only those methods that are sufficient to achieve the intent of the class, no more or no less.

Design Characteristics : Primitiveness

- Methods associated with a design class should be focused on accomplishing one service for the class. Once the service has been implemented with a method, the class should not provide another way to accomplish the same thing.

Design Characteristics : High Cohesion

- A cohesive design class has a small, focused set of responsibilities and single mindedly applies attributes and methods to implement those responsibilities.

Design Characteristics : Low Coupling

- Within the design model, it is necessary for design classes to collaborate with one another. Although this collaboration has to be kept to an acceptable minimum level.
- If the design model is highly coupled i.e. all design classes collaborate with all other design classes, the system is difficult to implement, to test and to maintain over long period of time.
- Hence, design classes within the subsystem should have only limited knowledge of the classes in other subsystem.

Design Characteristics : Low Coupling …

- This is called “Law of Demeter” which suggests that a method should only send messages to those in the neighboring classes.

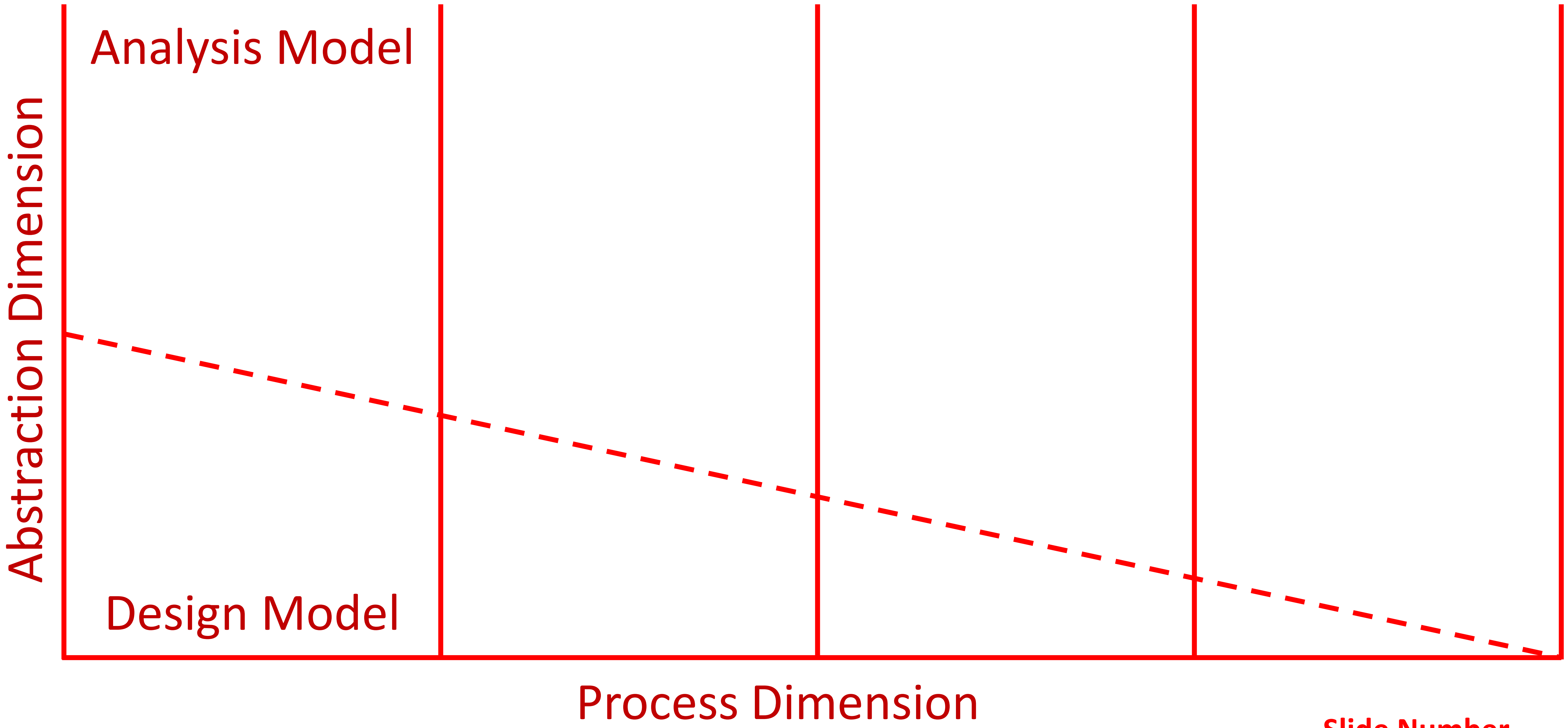
Design Characteristics : Low Coupling …

- This is called “Law of Demeter” which suggests that a method should only send messages to those in the neighboring classes.

Design Model

- The Design Model can be viewed in two dimensions. They are
    - Process Dimension
    - Abstraction Dimension

Design Model …



Design Model …

- In the above figure the dashed line separates the design and analysis models.
- In some cases, a clear distinction between analysis and design models is possible. And in other cases, the analysis model slowly blends with the design model and a clear distinction is less obvious.

Design Model : Process Dimension

- The Process Dimension indicates the evolution of the design model as design tasks are executed as part of the software process.

Design Model : Abstraction Dimension

- The Abstraction Dimension represents the level of details as each element of the analysis model is transformed into a design equivalent and then refined iteratively.

Data Design Elements

- Data Design or Data Architecting creates a model of data and /or information that is represented at a high level of abstraction i.e. catering the view point of the customer and / user of data. This data model is then refined into progressively more implementation specific representation that can be processed by the computer based system. In general, the architecture of the data will have a profound influence on the architecture of the software that must process it.

Data Design Elements …

- The structure of data has always been an integral part of software design. At the program component level, the design of data structures and the associated algorithms required to manipulate them is essential the creation of high quality applications.
- At the application level, the translation of a data model into a database is pivotal to achieving the business objectives of the system.

Data Design Elements …

- At the business level, the collection of information stored in disparate databases and reorganized into a “Data Warehouse” enables data mining or knowledge discovery that can have impact on the success of the business.

Architectural Design Elements

- The architectural design elements give us the overall view of the software. The architectural model is derived from following sources.
  - Information about the application domain for the software to be built.
  - Specific analysis model elements like data flow diagrams or analysis classes
  - Availability of architectural patterns.

Interface Design Elements

- The Interface Design for the software is very much equivalent to a set if detailed drawings and specification.
- It tells us that how information flows into and out of the system and how it is communicated among the components defined as part of the architecture.
- The elements of the interface are
  - The user interface
  - The external interfaces to other systems, devices, networks etc.
  - The internal interfaces between various design components.

Interface Design Elements …

- The interface element allows the software to communicate externally and enable internal communication and collaboration among the components of the software.

User Interface Design

- It incorporates
  - Aesthetic Elements
  - Ergonomic Elements
  - Technical Elements

User Interface Design : Aesthetic Elements

- Layout
- Color
- Graphics
- Interaction Mechanism

User Interface Design : Ergonomic Elements

- Information Layout and Placement
- Metaphors
- User Interface Navigation

User Interface Design : Technical Elements

- User Interface Pattern
- Reusable Components

External Interface Design

- It requires definitive information regarding the entity to which information is sent or received.
- The design of External Interfaces should incorporate error checking and security features.

Internal Interface Design

- It is closely aligned with component level design.
- Design realizations of analysis classes represent all operations and the messaging schemes to enable the communication and collaboration between operations in various classes.
- Each message must be designed to accommodate the requisite information transfer and the specific functional requirements of the operation that has been requested.

User Interface Design …

- Hence, we can say that, “An interface is a specifier for the externally visible i.e. public operations of a class, component or subsystems without specification of internal structure.”
- Thus “An interface is a set of operations that describe some part of the behaviour of a class and provide access to those operations.

Component Level Design Elements

- The component level design for software is equivalent to a set of detailed drawings and specifications for each element in a module.
- These drawing depicts and describes the internal details of each software component.
- To accomplish the above, the component level design defines data structures for all local data objects and algorithmic detail for all processing that occurs within a component and an interface that allows access to all component operations and behaviour.

Deployment Level Design Elements

- Deployment level design elements indicate how software functionality and subsystems will be allocated within the physical computing environment that will support the software.

Pattern Based Software Design

- The designers must have the ability to see patterns that characterize a problem and corresponding patterns that can be combined to create a solution. During the design process, a software engineer puts stress on the reuse of existing pattern (if they meet the current requirement) rather than creating a new one.
  - Describing a Design Pattern
  - Using Patterns in Design
  - Framework

Pattern Based Software Design : Describing a Design Pattern

- A description of design pattern requires design forces to be considered.
- Design forces describe the non functional requirements like ease of maintainability, portability etc associated with the software for which the pattern is to be applied. Design forces also define the constraints that may restrict the manner in which the design is to be implemented. Hence we can say that design forces describe the environment and conditions that must exist to make the design pattern applicable.

Pattern Based Software Design : Describing a Design Pattern …

- The pattern characteristics like classes, responsibilities and collaborations indicate the attributes of the design that may be adjusted enable the pattern to accommodate a variety of problems.

Pattern Based Software Design : Using Patterns in Design

- Design patterns can be used throughout the software design. Once the analysis model is developed, the designer can examine a detailed representation of the problem to be solved and the constraints that are imposed by the problem. The problem description is examined at various levels of abstraction to determine it is amenable to one or more types of design pattern.
  - Architectural Pattern
  - Design Pattern
  - Idioms

Architectural Pattern

- These patterns define the overall structure of the software, indicate the relationships among the subsystems and software components and define the rules for specifying relationships among the elements i.e. classes, packages, components, subsystems etc. of the architecture.

Design Pattern

- The pattern addresses a specific element of the design like an aggregation of components to solve the design problems, relationships among components or the mechanisms for effecting component to component communication.

Idioms i.e. Coding Pattern

- Idioms are generally referred as coding patterns.
- These language specific pattern generally implement an algorithmic element of a component, a specific interface protocol or a mechanism for communication among components.

Pattern Based Software Design : Framework

- Framework is referred as implementation specific skeletal infrastructure.
- A framework is not an architectural pattern.
- It is rather a skeleton with a collection of “Plug Points”.
- The “Plug Points” enable the designer to integrate problem specific classes or functionality within the skeleton.
- Framework can also be referred as “Co – Operating Classes”.