MALLA REDDY UNIVERSITY

MR22-1CS0146: OBJECT ORIENTED SOFTWARE ENGINEERING

II YEAR B.TECH

UNIT-III

- **Design Engineering :** Design process and Design quality, Design concepts, the design model.

- **Creating an architectural design :** **S**oftware architecture, Data design, Architectural styles and patterns, Architectural Design, conceptual model of UML, basic structural modeling, class diagrams, sequence diagrams, collaboration diagrams, use case diagrams, component diagrams.

Design Engineering

- **Design engineering** encompasses the **set of principles, concepts, and practices** that leads to the development of a high- quality system or product.

What is design?

- Design is what virtually every engineer wants to do.
- It is the place where creativity rules – customer's requirements, business needs, and technical considerations all come together in the formulation of a product or a system.

Why is it important?

- Design allows a software engineer to model the system or product that is to be built.
- Design is the place where software **quality is established.**

DESIGN PROCESS AND DESIGN QUALITY:

Software design is an iterative process through which requirements are translated into a “blueprint” for constructing the software.

Goals of design:

- The design must implement all of the explicit requirements contained in the analysis model .
- The design must be a readable, understandable guide.
- The design should provide a complete picture of the software

Quality guidelines:

- A design should exhibit an architecture that ,
  - **a.** has been created using recognizable architectural styles or patterns
  - **b.** is composed of components that exhibit good design characteristics and
  - **c.** can be implemented in an evolutionary fashion, thereby facilitating implementation and testing.
- A design should be modular.

- lead to data structures that are appropriate for the classes to be implemented .
- lead to components that exhibit independent functional characteristics.
- lead to interface that reduce the complexity of connections between components and with the external environment.
- should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
- should be represented using a notation that effectively communicates its meaning.

Quality attributes: (FURPS)

- ***Functionality***
- ***Usability***
- ***Reliability***
- ***Performance***

DESIGN CONCEPTS:

Fundamental software design concepts provide the necessary framework for “getting it right.”

1. Abstraction: Many levels of abstraction are there,

- At the highest level of abstraction, a solution is stated in broad terms using the language of the problem environment.
- At lower levels of abstraction, a more detailed description of the solution is provided.
- A ***procedural abstraction*** refers to a sequence of instructions that have a specific and limited function.

2. Architecture:

- the overall structure of the software and the ways in which that structure provides conceptual integrity for a system
  - ***Structured models*** - organized collection of program components.
  - ***Framework models -*** increase the level of design abstraction.
  - ***Dynamic models -*** address the behavioral aspects of the program architecture, indicating how the structure or system configuration may change as a function external events.
  - ***Process models -*** focus on the design of the

3. Patterns:

- a design pattern describes a design structure that solves a particular design within a specific context and amid “forces” that may have an impact on the manner in which the pattern is applied and used.

- The intent of each design pattern is to provide a description that enables a designer to determine

1) Whether the pattern is capable to the current work,

2) Whether the pattern can be reused,

3) Whether the pattern can serve as a guide for developing a similar, but functionally or structurally

4. Modularity

- software is divided into separately named and addressable components, sometimes called ***modules*** that are integrated to satisfy problem requirements.

5. Information Hiding:

- information contained within a module is inaccessible to other modules that have no need for such information.

6. **Functional Independence:**

- direct outgrowth of modularity and the concepts of abstraction and information hiding.
- Independence is assessed using two qualitative criteria: cohesion and coupling.
  - *Cohesion* is an indication of the relative functional strength of a module.
  - *Coupling* is an indication of the relative

7. Refinement:

- A program is development by successively refining levels of procedural detail.
- Refinement is actually a process of elaboration.

8. Refactoring:

- Refactoring is a reorganization technique that simplifies the design of a component without changing its function or behavior.

9. Design classes:

- The software team must define a set of design classes that ,
  - 1. Refine the analysis classes by providing design detail that will enable the classes to be implemented, and
  - 2. Create a new set of design classes that implement

Five different types of design classes ,

- **User interface classes:** define all abstractions that are necessary for human computer interaction.
- **Business domain classes:** are often refinements of the analysis classes defined earlier.
- **Process classes** implement lower – level business abstractions required to fully manage the business domain classes.
- **Persistent classes** represent data stores that will persist beyond the execution of the software.
- **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.

Four characteristics of a well- formed design class,

- Complete and sufficient

THE DESIGN MODEL:

i. Data design elements:

- creates a model of data and/or information that is represented at a high level of abstraction.
- The structure of data has always been an important part of software design.
  - At the **program component level**
  - At **the application level**
  - At the **business leve**l

ii. Architectural design elements:

- The *architectural design* for software is the equivalent to the floor plan of a house. The architectural model is derived from three sources.
  - a. Information about the application domain for the software to be built.
  - b. Specific analysis model elements, and
  - c. The availability of architectural patterns

iii. Interface design elements:

- equivalent to a set of detailed drawings for the doors, windows, and external utilities of a house.
- There are 3 important elements of interface design:
  - a. The user interface(UI);
  - b. External interfaces to other systems, devices, networks, or other produces or consumers of information; and
  - c. Internal interfaces between various design components.

iv. Component- level design elements:

- It is equivalent to a set of detailed drawings.
- The component-level design for software fully describes the internal detail of each software component.

v. Deployment-level design elements:

ARCHITECTURAL DESIGN

SOFTWARE ARCHITECTURE:

What Is Architecture?

- Architectural design represents the structure of data and program components that are required to build a computer-based system.
- The design of software architecture considers two levels of the design pyramid
  - - data design
  - - architectural design.

Why Is Architecture Important?

- Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system.
- The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.

DATA DESIGN:

- translates data objects as part of the analysis model into data structures at the software component level and, when necessary, a database architecture at the application level.

Data design at the Architectural Level:

- The challenge for a business has been to extract useful information from this data environment, particularly when the information desired is cross functional.

2. **Data design at the Component Level**:

- focuses on the representation of data structures that are directly accessed by one or more software components

The following set of principles for data specification:

1. The systematic analysis principles applied to function and behavior should also be applied to data.
2. All data structures and the operations to be performed on each should be identified.
3. A data dictionary should be established and used to define both data and program design.
4. Low-level data design decisions should be deferred until late in the design process.
5. The representation of data structure should be known only to those modules that must make direct use of the data contained within the structure.
6. A library of useful data structures and the operations that may be applied to them should be developed.
7. A software design and programming language should support the specification and realization of abstract data types.

ARCHITECTURAL STYLES AND PATTERNS:

- The software that is built for computer-based systems also exhibits one of many architectural styles.
- Each style describes a system category that encompasses
  (1) A set of ***components***
  (2) A set of ***connectors***
  (3) ***Constraints***
  (4) ***Semantic models***

An ***architectural pattern,*** imposes a transformation the design of architecture. However, a pattern differs from a style in a number of fundamental ways:

(1) The scope of a pattern is less broad, focusing on one aspect of the architecture rather than the architecture in its entirety.

(2) A pattern imposes a rule on the architecture, describing how the software will handle some aspect of its functionality at the infrastructure level

1. A Brief Taxonomy of Styles and Patterns
Data-centered architectures:

Data-flow architectures:

Call and return architectures:

- ***Main program/subprogram architectures.***
- ***Remote procedure call architectures.***

Object-oriented architectures:

- The components of a system encapsulate data and the operations that must be applied to manipulate the data.

Layered architectures:

2. Architectural Patterns:

- The architectural patterns for software define a specific approach for handling some behavioral characteristics of the system,
  - **Concurrency**
    - *operating system process management* pattern
    - *task scheduler* pattern
  - **Persistence**
    - a ***database management system*** pattern
    - an ***application level persistence*** pattern
  - **Distribution**
- **Organization and Refinement:**

ARCHITECTURAL DESIGN:

i. Representing the System in Context:

ii. Defining Archetypes:

- An archetype is a class or pattern that represents a core abstraction that is critical to the design of architecture for the target system.

iii. Refining the Architecture into Components:

- As the architecture is refined into components, the structure of the system begins to emerge.

iv. Describing Instantiations of the System:

A Conceptual Model of UML :

- UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

Object-Oriented Concepts

- UML can be described as the successor of object-oriented (OO) analysis and design.
- Following are some fundamental concepts of the object-oriented world,
- **Objects** − Objects represent an entity and the basic building block.
- **Class** − Class is the blue print of an object.
- **Abstraction** − Abstraction represents the behavior of an real world entity.
- **Encapsulation** − Encapsulation is the mechanism of binding the data together and hiding them from the outside world.
- **Inheritance** − Inheritance is the mechanism of making

OO Analysis and Design:

- OO can be defined as an investigation and to be more specific, it is the investigation of objects. Design means collaboration of identified objects.

- The purpose of OO analysis and design can described as –
  - Identifying the objects of a system.
  - Identifying their relationships.
  - Making a design, which can be converted to executables using OO languages.

- There are three basic steps where the OO concepts are applied and implemented. The steps can be defined as,

OO Analysis → OO Design → OO implementation using OO languages

Basic Structural Modeling:

- Structural modeling captures the static features of a system. They consist of the following −
  - Classes diagrams
  - Objects diagrams
  - Deployment diagrams
  - Package diagrams
  - Composite structure diagram
  - Component diagram
- Structural model represents the framework for the system and this framework is the place where all other components exist.

Class Diagram:

- Class diagram is a static diagram that shows a collection of classes, interfaces, associations, collaborations, and constraints.

Purpose of Class Diagrams

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.

The Sequence Diagram:

- The sequence diagram has four objects (Customer, Order, SpecialOrder and NormalOrder).

The Collaboration Diagram :

Use Case Diagram:

- It is used to represent the dynamic behavior of a system.
- It encapsulates the system's functionality by incorporating use cases, actors, and their relationships.

Purpose of Use Case Diagrams

1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

Some rules that must be followed while drawing a use case diagram:

1. A pertinent and meaningful name should be assigned to the actor or a use case of a system.
2. The communication of an actor with a use case must be defined in an understandable way.
3. Specified notations to be used as and when required.

Component diagram:

- used to model the physical aspects of a system.
- Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.

Purpose of Component Diagrams

- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.