UNIT-4

Chapter-9

The Object Oriented Design process and Design Axioms.

Introduction:-

Objects discovered during analysis serves as a framework for design.

The class's attributes, methods and associations identified during analysis must be designed for implementation.

New classes must be introduced to store intermediate results during program execution.

Analysis → Physical entities → represents tangible elements of the business.

During design phase we elevate the model into logical entities.

Here we focus on view and access classes.

Good design usually simplifies the implementation and maintenance of a project.

The object oriented design process:-

O-O Design process consist of following activities

1.) Apply design axioms to design classes, attributes, methods & associations, structures & protocols.

1.1.) Refine and compute static UML class diagram by

1.1.1) Refine attributes

1.1.2) Refine associations

1.1.3) Refine class hierarchy and design with inheritance.

1.1.4) Design methods & protocols by UML activity diagram.

1.2) Iterate & refine again.

2.) Design the access layer

2.1) Create mirror classes:-

For every business class identified and created, create one access class. (mirror class).

Ex: Business cl: Access class - C1, C2, C3

access class - AC1, AC2, AC3.

2.2) Simplify classes and their relationship:

(1) Main goal - Eliminate redundant classes and structures.

→ Redundant classes ⇒ Do not keep 2 classes that perform same activities. Keep one and eliminate other.

→ Method classes ⇒ See methods can be eliminated or combined with existing classes.

2.3) Identify access layer class relationship.

2.4) Design the view layer classes:

1.) Design macro level user interface → identify view layer objects.

2.) Design micro level user interface →

1.) Design view layer objects by applying design axioms & corollaries.

2.) Build a prototype of view layer interface.

3.) Test usability and user satisfaction.

4.) Iterate and refine.

2.5) Iterate and refine the whole design.

Object Oriented Design axioms:-

Axiom => a fundamental truth observed to be valid and for which there is no example.
=> They cannot be proven or derived.

Theorem => May not be self evident but can be Proven from accepted axioms.

Corollary => Follows from an axiom or another Proposition that has been Proven.

Axiom1 => Deals with Relationship b/w system components. (Such as class, requirement & s/w components).

Axiom 2 => Deals with complexity of design.

Axiom 1:- The independence axiom.
↳ Maintain the independence of components.

Axiom 2:- The information axiom
↳ Minimize the information content of the design.

=> Axiom 1 States that during design process, as we go from requirement and use case to a system component, each component must satisfy that requirement without affecting other requirement.
Ex: Refrigerator door design.

=> Axiom 2 → Concerned with simplicity.
→ General rule known as Occam's razor.
→ minimum amount of complexity and maximum simplicity.

Corollaries:-

Coupling => degree of connection blw objects.

Cohesion => degree of internal consistency or unity within an object. <=> Method cohesion, Class cohesion, Inheritance cohesion

-> A good OOAD System should have low coupling and high cohesion, meaning that objects are loosely coupled & highly cohesive.

---> From the 2 axioms, corollaries are derived as a direct consequence.

-> Also called as design rules.

Corollary 1:- Uncoupled design with less information content:-

↳ High cohesion, low coupling

Corollary 2:- Single purpose:-

↳ Each class must have single & clearly defined Purpose.

Corollary 3:- Large number of simple classes:-

↳ Keeping class simple allows reusability.

Corollary 4:- Strong mapping:-

↳ Strong association blw analysis & design object.

Corollary 5:- Standardization:-

↳ By designing interchangable components and reusing existing class or components.

Corollary 6:- Design with Inheritance:-

↳ Superclass - Subclass structure

Design Patterns:-

Design Patterns are devices that allows system to share knowledge about this design by describing commonly recurring structures of communicating components that solve a general design problem within a particular context.

Design Pattern example created by Kurotsuchi:-

1.) Pattern Name: Facade

2.) Rationale and Motivation:

→ Accessing a large number of modules much simpler by providing an additional interface layer.

→ Done by creating small collection of classes that have a single class used to access them.

3.) Classes:-

→ Atleast 4 to 5 classes are required to involve in facade system.

4.) Advantages / Disadvantages:

↳ Make the interfacing b/w many modules or classes more manageable.

May lose some functionality contained in the lower level of classes.

chapter - 10
Designing classes

Introduction:

A class designer needs to know ~~that~~ the specifics of a class and to know how the class interacts with other classes.

Object oriented design Philosophy:-

→ in O.O approach, classes organize related properties into units.

→ Important activity in designing an application is merging set of classes that work together to provide the functionality.

→ Many O-O programming languages has several built in class libraries.

→ Main goal: reuse rather than creating new.

→ Use design axioms.

UML Object Constraint language:-

UML → A graphical language with a set of rules and semantics.

OCL → specification language uses simple logic for specifying the properties of a system.

UML → UML modeling require expression.

OCL → In OCL expressions are stated as strings.

↓

Expressions are meant to work on set of values when applicable.

↓

Left most element

1.) Item . Selector => Ex: John. age

Item → left most so Expression of object

Selector → name of an attribute in the Item

2.) Item . Selector [qualifier - value].
Ex: John. Phone [2].

3.) Set -> Select (boolean - expression)
Ex: Employee → salary 75000

Class Visibility : Private, Public & Protected Protocols :-

Private → hidden from other objects
Public → made available to other objects.

Public protocol => define the functionality and external messages of an object.

Private Protocol => define the implementation of object.

-> It's important to define the public protocol b/w the associated classes in the application.

Protected => set of methods uses only internally to message itself. (subclasses can use the method in addition to class itself).

Private => Messages should not be sent from other objects, it is accessible only to operations of that class.

Public => Accesible to all classes.

1.) Private and Protected Protocol layers : Internal :-

→ Define the Implementation of the object.
→ Corollary 1

2.) Public Protocol layer : External :-

→ Define the functionality of the object.
→ Points to remember, when designing class protocols
1.) Good design allows for Polymorphism.
2.) not all protocols should be public; again apply design axioms & corollaries.

Designing classes: Refining attributes :-

Main goal → To refine the attributes identified on analysis Phase (or) to add new attributes that elevate the system into Implementation.

Attribute types:
1.) Single-valued
2.) Multiplicity or multivalued attributes.
3.) Reference to another object or Instance connection.

Instance connection => These attributes are required to provide the mapping needed by an object to fulfill its responsibilities.

UML attribute Representation :-

Visibility name : type-expression = Initial-value

Visibility:-

+ => Public
# => Protected
- => Private

Type expression:
Implementation type of an attribute (names for datatype variables).

Initial value:
-> Initial value of newly created object.
-> Initial value is optional.

(!) UML style guidelines recommend beginning attribute names with a lower-case letter.

Ex: id : string
+id : string
+id : Partnra

Designing methods and Protocols:-
Main goal => Specify the algorithm for methods identified.
Designing methods => by UML activity diagram
↓
they can be converted to programming language manually or in automated fashion.

Types of methods:

1.) Constructor → Method that creates instances of class
2.) Destructor → Method that destroys instances
3.) Conversion method → Method that converts value from one unit to another.
4.) Copy method → Copies the contents of one instance to another instance.
5.) Attribute Set → Sets the value of one or more attribute
6.) Attribute get → Returns the value of one or more attri
7.) I/o methods → Provide & receive data to & from a dev
8.) Domain Specific → Specific to the application.

Corollary 1 → Maximize cohesiveness to improve coupling.

Five rules:

1.) Looks messy, bad design
2.) Too complex, bad design
3.) Too big, bad design
4.) People don't like, bad design
5.) Doesn't work, bad design.

Design Issues: Avoiding Design Pitfalls:-

Goal → Maximum reuse rather than create new.
→ A meaningful class definition starts out simple.
→ But when changes are made class becomes larger & larger, which makes the class harder to state concisely.

→ This happens when we make incremental changes to an existing class.
→ To avoid complexity use axioms & corollaries (2).
→ But this problem can be detected early.

↓

→ There were bugs because the internal state of an object is too hard to track.
→ Some possible actions to solve this are
1.) Keep a careful eye on class design & if an object loses focus, we need to modify the design. (corollary 2).
2.) Move some functions into new class. (corollary 1)
3.) Breakup the class into two or more classes (Corollary 3).
4.) Rethink the class definition based on experience gained.

UML operation Presentation:-

Visibility name : (Parameter-list) : return-type-expression

Ex: + getName() : aName.
+ getAccountNumber (account:type) : accountNumber

Packages and managing classes:-
→ Packages groups & manages the modeling element, such as classes, their association & their structures.
→ Package contains other packages and ordinary model elements.

→ Entire system can be thought of as a single high-level subsystem Package.

→ Packages own model elements and model fragements, so they can be used by CASE tools as the basic storage and access control.

→ The keywords subsystem and model indicate that the package is a meta-model subsystem or model