Dhananjay Porwal **EN19CS301110**

MEDICAPS UNIVERSITY, INDORE

Computer Science & Engg. Department

Session Jan to May- 2022

<u>Course</u>: Object Oriented Analysis & Design **<u>Course Code:</u>** CS3CO15
<u>Year:</u> III yr **Semester :** Even(VI)
<u>Date of Assignment:</u> 3 Feb 2022 **<u>Date of Submission</u>:** 11 Feb 2022

1. Why Software Is Inherently Complex.

The complexity of software is an essential property not an accidental one. The inherent complexity derives from four elements; the complexity of the problem domain, the difficultly of managing the developmental process, the flexibility possible through software and the problems of characterizing the behavior of discrete systems.

<u>1.The complexity of the problem domain</u>

- Complex requirements
- Decay of system

The first reason has to do with the relationship between the application domains for which software systems are being constructed and the people who develop them.
The second reason is the complexity of the software development process. Complex software intensive systems cannot be developed by single individuals. They require teams of developers.
The Difficulty of Managing the Development Process

- Management problems
- Need of simplicity

<u>2.The flexibility possible through software</u>

- Software is flexible and expressive and thus encourages highly demanding requirements, which in turn lead to complex implementations which are difficult to assess.

The third reason is the danger of flexibility. Flexibility leads to an attitude where developers develop system components themselves rather than purchasing them from somewhere else. Software offers the ultimate flexibility. It is highly unusual for a construction firm to build an on site steel mill to forge (create with hammer) custom girders (beams) for a new building. Construction industry has standards for quality of row materials, few such standards exist in the software industry.

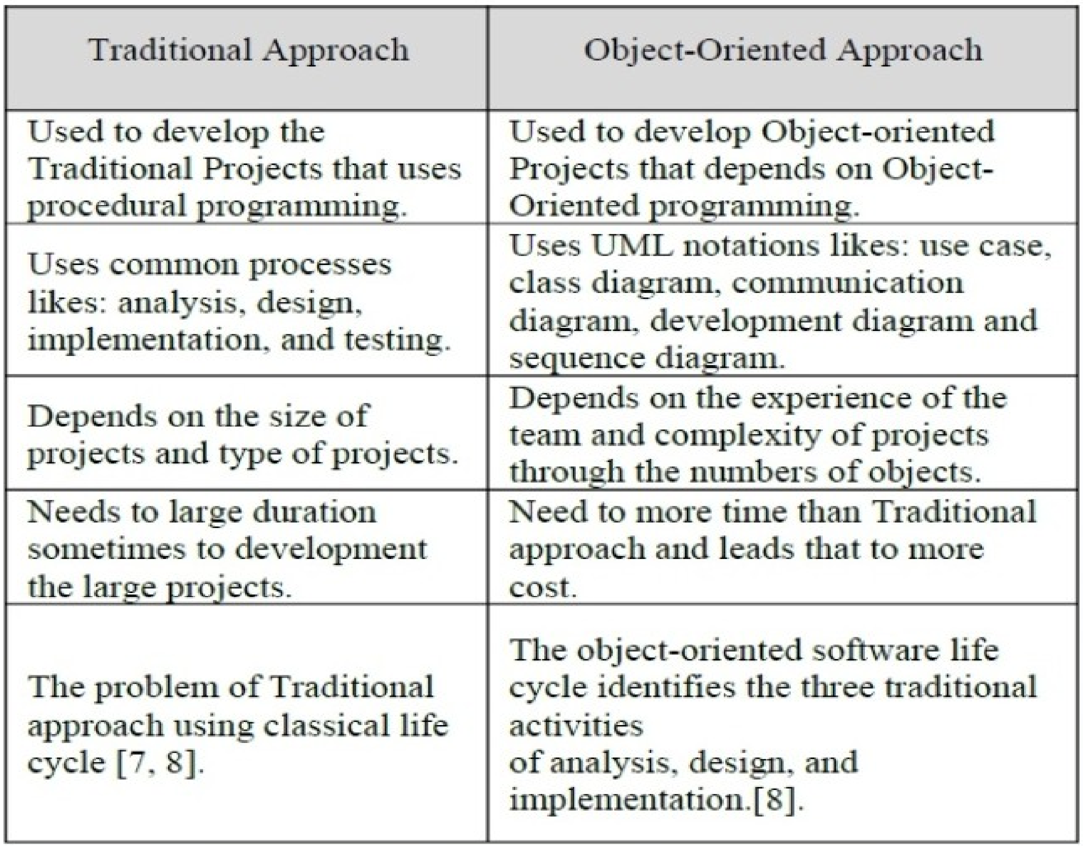
<u>3.The problem of characterizing the behaviour of discrete systems</u>

- Numerous possible states
- Difficult to express all states

The final reason for complexity according to Booch is related to the difficulty in describing the behaviour of software systems. Humans are capable of describing the static structure and properties of complex systems if they are properly decomposed, but have problems in describing their behaviour. This is because to describe behaviour, it is not sufficient to list the properties of the system. It is also necessary to describe the sequence of the values that these properties take over time.

Dhananjay Porwal **EN19CS301110**

2. Give the difference between Traditional and Object Oriented Approach(methods)?



Traditional Approach	Object-Oriented Approach
Used to develop the Traditional Projects that uses procedural programming.	Used to develop Object-oriented Projects that depends on Object-Oriented programming.
Uses common processes likes: analysis, design, implementation, and testing.	Uses UML notations likes: use case, class diagram, communication diagram, development diagram and sequence diagram.
Depends on the size of projects and type of projects.	Depends on the experience of the team and complexity of projects through the numbers of objects.
Needs to large duration sometimes to development the large projects.	Need to more time than Traditional approach and leads that to more cost.
The problem of Traditional approach using classical life cycle [7, 8].	The object-oriented software life cycle identifies the three traditional activities of analysis, design, and implementation.[8].

3. Define the concept object orientation? Also give principles of oo system.

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behaviour.
OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well-suited for programs that are large, complex and actively updated or maintained.
Once an object is known, it is labeled with a class of objects that defines the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method.

The structure, or building blocks, of object-oriented programming include the following:
Classes are user-defined data types that act as the blueprint for individual objects, attributes and methods.
Objects are instances of a class created with specifically defined data. Objects can correspond to real-world objects or an abstract entity. When class is defined initially, the description is the only object that is defined.
Methods are functions that are defined inside a class that describe the behaviors of an object. Each method contained in class definitions starts with a reference to an instance object. Additionally, the subroutines contained in an object are called instance methods. Programmers use methods for reusability or keeping functionality encapsulated inside one object at a time.

Attributes are defined in the class template and represent the state of an object. Objects will have data stored in the attributes field. Class attributes belong to the class itself.

Object-oriented programming is based on the following principles:
Encapsulation. This principle states that all important information is contained inside an object and only select information is exposed. The implementation and state of each object are privately held inside a defined class. Other objects do not have access to this class or the authority to make changes. They are only able to call a list of public functions or methods. This characteristic of data hiding provides greater program security and avoids unintended data corruption.
Abstraction. Objects only reveal internal mechanisms that are relevant for the use of other objects, hiding any unnecessary implementation code. The derived class can have its functionality extended. This concept can help developers more easily make additional changes or additions over time.
Inheritance. Classes can reuse code from other classes. Relationships and subclasses between objects can be assigned, enabling developers to reuse common logic while still maintaining a unique hierarchy. This property of OOP forces a more thorough data analysis, reduces development time and ensures a higher level of accuracy.
Polymorphism. Objects are designed to share behaviors and they can take on more than one form. The program will determine which meaning or usage is necessary for each execution of that object from a parent class, reducing the need to duplicate code. A child class is then created, which extends the functionality of the parent class. Polymorphism allows different types of objects to pass through the same interface.

4. Explain RUP with its phases?

Rational Unified Process (RUP) is a software development process for object-oriented models. It is also known as the Unified Process Model. It is created by Rational corporation and is designed and documented using UML (Unified Modeling Language). This process is included in IBM Rational Method Composer (RMC) product. IBM (International Business Machine Corporation) allows us to customize, design, and personalize the unified process.
RUP is proposed by Ivar Jacobson, Grady Bootch, and James Rambaugh. Some characteristics of RUP include use-case driven, Iterative (repetition of the process), and Incremental (increase in value) by nature, delivered online using web technology, can be customized or tailored in modular and electronic form, etc. RUP reduces unexpected development costs and prevents wastage of resources.

Phases of RUP :
There are total five phases of life cycle of RUP:

- Inception –

Communication and planning are main.
Identifies Scope of the project using use-case model allowing managers to estimate costs and time required.
Customers requirements are identified and then it becomes easy to make a plan of the project.
Project plan, Project goal, risks, use-case model, Project description, are made.
Project is checked against the milestone criteria and if it couldn't pass these criteria then project can be either cancelled or redesigned.

- Elaboration –

Planning and modeling are main.
Detailed evaluation, development plan is carried out and diminish the risks.

Revise or redefine use-case model (approx. 80%), business case, risks.
Again, checked against milestone criteria and if it couldn't pass these criteria then again project can be cancelled or redesigned.
Executable architecture baseline.

- Construction –

Project is developed and completed.
System or source code is created and then testing is done.
Coding takes place.

- Transition –

Final project is released to public.
Transit the project from development into production.
Update project documentation.
Beta testing is conducted.
Defects are removed from project based on feedback from public.

- Production –

Final phase of the model.
Project is maintained and updated accordingly.

5. Write short note:

1. Building Blocks

UML is composed of three main building blocks, i.e., things, relationships, and diagrams. Building blocks generate one complete UML model diagram by rotating around several different blocks. It plays an essential role in developing UML diagrams. The basic UML building blocks are enlisted below:

- Things
- Relationships
- Diagrams

➢ Things

Anything that is a real world entity or object is termed as things. It can be divided into several different categories:

- Structural things

Nouns that depicts the static behavior of a model is termed as structural things. They display the physical and conceptual components. They include class, object, interface, node, collaboration, component, and a use case.

Class: A Class is a set of identical things that outlines the functionality and properties of an object. It also represents the abstract class whose functionalities are not defined. Its notation is as follows;
Object:: An individual that describes the behavior and the functions of a system. The notation of the object is similar to that of the class; the only difference is that the object name is always underlined and its notation is given below;
Interface: A set of operations that describes the functionality of a class, which is implemented whenever an interface is implemented.
Collaboration: It represents the interaction between things that is done to meet the goal. It is symbolized as a dotted ellipse with its name written inside it.

Use case: Use case is the core concept of object-oriented modeling. It portrays a set of actions executed by a system to achieve the goal.

Actor: It comes under the use case diagrams. It is an object that interacts with the system, for example, a user.

- Behavioral Things

They are the verbs that encompass the dynamic parts of a model. It depicts the behavior of a system. They involve state machine, activity diagram, interaction diagram, grouping things, annotation things

State Machine: It defines a sequence of states that an entity goes through in the software development lifecycle. It keeps a record of several distinct states of a system component.

Activity Diagram: It portrays all the activities accomplished by different entities of a system. It is represented the same as that of a state machine diagram. It consists of an initial state, final state, a decision box, and an action notation.
Interaction Diagram: It is used to envision the flow of messages between several components in a system.

- Grouping Things

It is a method that together binds the elements of the UML model. In UML, the package is the only thing, which is used for grouping.
Package: Package is the only thing that is available for grouping behavioral and structural things.

- Annotation Things

It is a mechanism that captures the remarks, descriptions, and comments of UML model elements. In UML, a note is the only Annotational thing.

➢ Relationships

It illustrates the meaningful connections between things. It shows the association between the entities and defines the functionality of an application. There are four types of relationships given below:
Dependency: Dependency is a kind of relationship in which a change in target element affects the source element, or simply we can say the source element is dependent on the target element. It is one of the most important notations in UML. It depicts the dependency from one entity to another.

It is denoted by a dotted line followed by an arrow at one side as shown below,
UML-Building Blocks
Association: A set of links that associates the entities to the UML model. It tells how many elements are actually taking part in forming that relationship.

It is denoted by a dotted line with arrowheads on both sides to describe the relationship with the element on both sides.
UML-Building Blocks
Generalization: It portrays the relationship between a general thing (a parent class or superclass) and a specific kind of that thing (a child class or subclass). It is used to describe the concept of inheritance.

It is denoted by a straight line followed by an empty arrowhead at one side.
UML-Building Blocks
Realization: It is a semantic kind of relationship between two things, where one defines the behavior to be carried out, and the other one implements the mentioned behavior. It exists in interfaces.
It is denoted by a dotted line with an empty arrowhead at one side.

➢ Diagrams

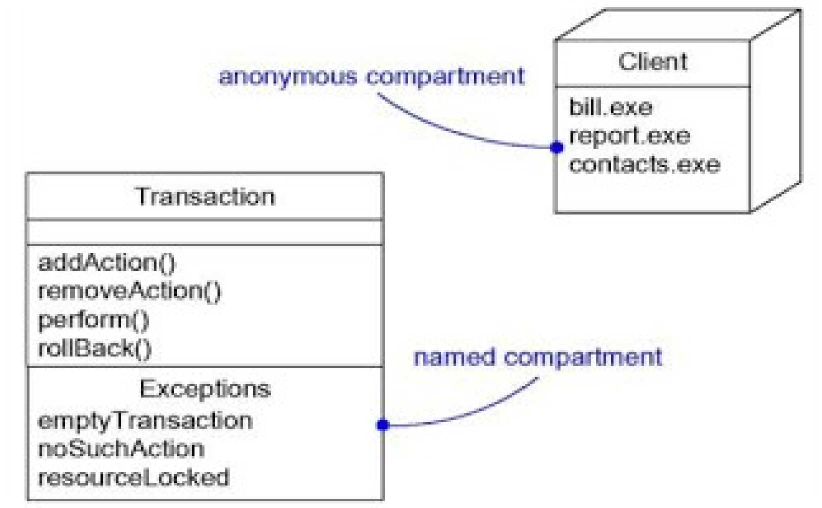
The diagrams are the graphical implementation of the models that incorporate symbols and text. Each symbol has a different meaning in the context of the UML diagram. There are thirteen different types of UML diagrams that are available in UML 2.0, such that each diagram has its own set of a symbol.

UML diagrams are classified into three categories that are given below:
Structural Diagram: It represents the static view of a system by portraying the structure of a system. It shows several objects residing in the system. Following are the structural diagrams given below: Class diagram , Object diagram , Package diagram , Component diagram , Deployment diagram
Behavioral Diagram: It depicts the behavioral features of a system. It deals with dynamic parts of the system. It encompasses the following diagrams: Activity diagram , State machine diagram , Use case diagram
Interaction diagram: It is a subset of behavioral diagrams. It depicts the interaction between two objects and the data flow between them. Following are the several interaction diagrams in UML: Timing diagram , Sequence diagram , Collaboration diagram

2. Common Mechanisms.

UML provides the basic notations to visualize, specify, construct and document the artifacts of a software intensive system. Sometimes, the user might need to represent information through notations which are not available in UML.
In such circumstances, we can use the extensibility mechanisms like stereotypes, tagged values and constraints which are a part of common mechanisms in UML.
UML supports an annotational thing known as a note. A note is used to provide comments or reviews or also for documentation.
Stereotypes, tagged values and constraints are the mechanisms provided by the UML to extend the building blocks, the properties and the semantics. For example we can represent routers and hubs using stereotyped nodes.
We can maintain version numbers and the author name by using tagged values. We can specify certain range of values or other types of rules using constraints.

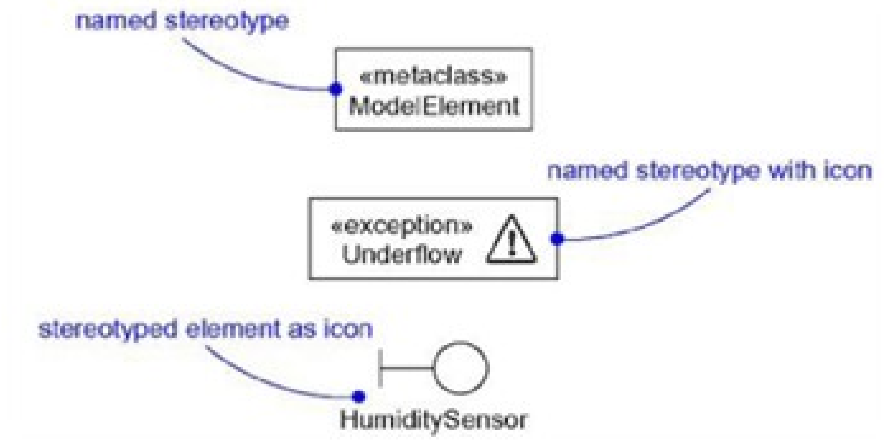
Other Adornments
Adornments are textual or graphical items that are added to the elements basic notation to specify extra information. When using UML, always start with the basic notation of elements and add adornments to specify new information. Example: adornments of an association.



Stereotypes

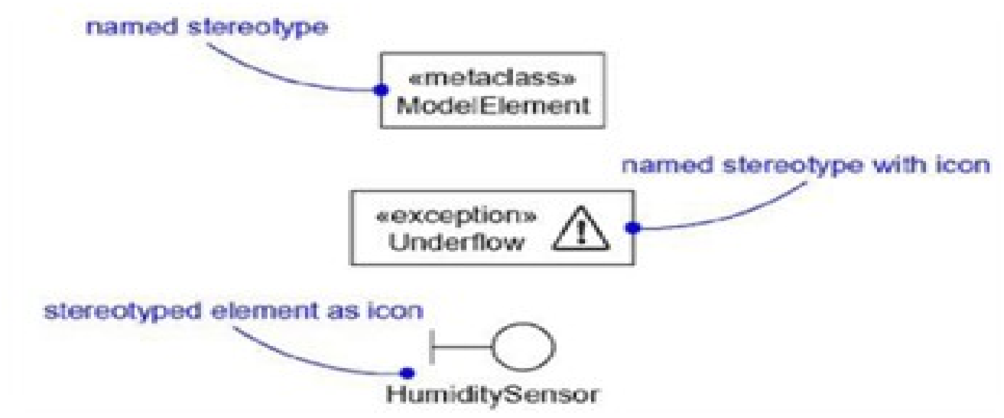
UML provides basic notations for structural things, behavioral things, grouping things and annotational things. Sometimes we may need to extend i.e., create new vocabulary and look like primitive building blocks.

Stereotypes allows us to create new building blocks. Using stereotypes we can use the basic elements but with special properties, semantics and notation. Stereotypes are rendered as text inside guillemets(<< >>) or we can create new icons for stereotypes.



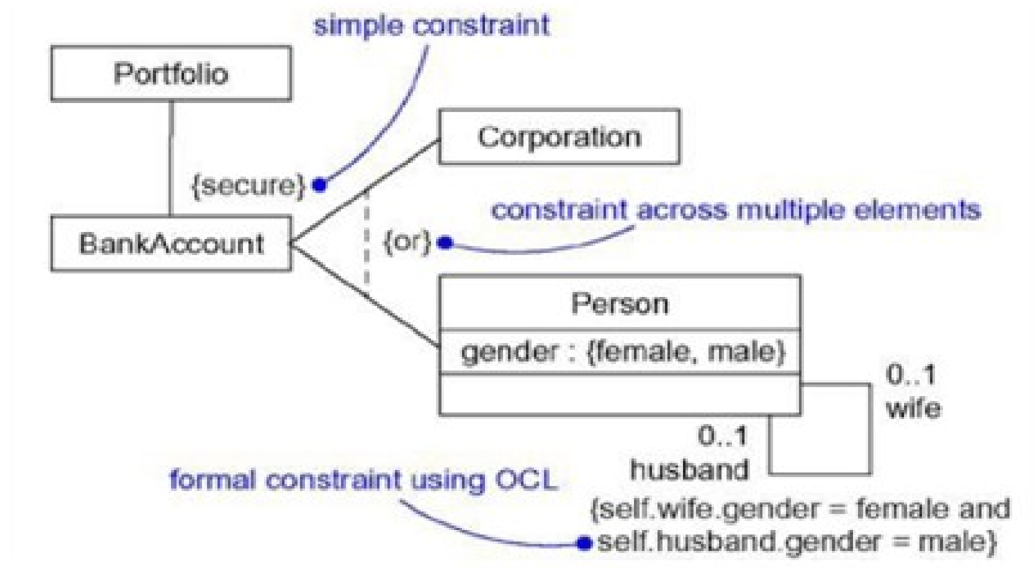
Tagged Values

Every element in UML has its own properties. For example a class has its own attributes and operations. Using tagged values, we can represent new properties also called as metadata. These properties apply to the element itself rather than its instance. Tagged values are enclosed in braces { } and are written under the element name.



Constraints

A constraint is used to add new semantics or change existing rules. The constraints specify rules that must be followed by the elements in the model. Represented as text inside braces { } and placed near to the associated element.


simple constraint
Portfolio
Corporation
{secure}
constraint across multiple elements
BankAccount
{or}
Person
gender : {female, male}
0..1
wife
0..1
husband
formal constraint using OCL
{self.wife.gender = female and
self.husband.gender = male}