UNIT II

METHODOLOGY AND UML
• Overview of methodologies:
  – OMT
  – Booch methodology
  – Jacobson methodology
  – Unified Approach

• UML:
  – Static and Dynamic Modelling
  – UML diagrams.
• Overview of methodologies:
  – OMT
  – Booch methodology
  – Jacobson methodology
  – Unified Approach

• UML:
  – Static and Dynamic Modelling
  – UML diagrams.
What is Object Oriented methodologies?

- Is a set of models, methods and rules for developing systems
- Is important for analysis and design as like object oriented programming concepts
Models:-
✓ description about the current and proposed system
✓ Used in all phase of software life cycle
✓ Easy to understand

Methods:-
Different methods developed in 1980’s – 1990’s are listed below
<table>
<thead>
<tr>
<th>Year</th>
<th>Created / Developed by</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1986</td>
<td>Booch</td>
<td>OO design concept BOOCH METHOD</td>
</tr>
<tr>
<td>1987</td>
<td>Sally &amp; Steve</td>
<td>Recursive design concept</td>
</tr>
<tr>
<td>1989</td>
<td>Beck &amp; Cunningham</td>
<td>CRC Card (Class – Responsibility – Collaboration card)</td>
</tr>
<tr>
<td>1990</td>
<td>Brock Weiner</td>
<td>Responsibility driven design</td>
</tr>
<tr>
<td>1991</td>
<td>Rumbaugh</td>
<td>OMT Object Modeling Technique</td>
</tr>
<tr>
<td></td>
<td>Pela</td>
<td>Load light weight and prototype oriented approach</td>
</tr>
<tr>
<td>1994</td>
<td>Jacobson</td>
<td>Use case and OOSE(Object Oriented Software Engineering)</td>
</tr>
</tbody>
</table>

All methodologies are same and related to one another but with slight difference
• Overview of methodologies:
  – OMT
  – Booch methodology
  – Jacobson methodology
  – Unified Approach

• UML:
  – Static and Dynamic Modelling
  – UML diagrams.
Rumbaugh – OMT

- Describes a method for analysis, design and implementation of a system using an OO techniques
- It is the fast approach for identifying the different object in a system
- Details such as class, attributes, methods, inheritance and association can be expressed easily
- It consists of four phases – performed iteratively
1. Analysis
2. System design
3. Object design
4. Implementation
1. **Analysis:** Results are object, dynamic and functional models

2. **System design:** Results are a structure of basic architecture of the system along with the high-level strategy decision

3. **Object design:** produce design document, consisting of detailed objects static, dynamic and functional models

4. **Implementation:** produce reusable, extensible and robust code
OMT separates modelling into three different parts

1. Object model
2. Dynamic model
3. Functional model
1. **Object Model:**- presented by object model and data dictionary, eg., different classes and its relations

2. **Dynamic model:**- presented by state diagram and event flow diagram eg., different events and dynamic objects

3. **Functional model:**- presented by dataflow diagram and constraints eg., different process descriptions and consumer-producer relationships

12/26/2012
1. Object Model:-

- Describes the structure of object in a system, their identity, relationships to other objects, attributes and operations
- It is represented graphically by object diagram
- Object diagram contains classes connected by association lines
- Each represent a set of individual object
- Following diagram is an example for object diagram/ model
2. Dynamic Model:-

- State the different states, transitions, action and event flow
- To represent these state transition diagram is used which is a collection of different states and events
- Each state receive one or more events, transitioned to next state
- Next state depends on the current state as well as the event.
- Following diagram is an example for state diagram/ dynamic model
No Account has been selected

Nothing is selected

Selected checking or saving account

Select transaction types (withdraw, deposit, transfer)

Account has been selected

Select checking Account

Enter the amount

Confirmation

Transition Lines

State
3. Functional Model

- Explain the different data flow between a process in a system by means of DFD (Data Flow Diagram)
- Give a simple description about process without focusing on the detail of system
- DFD mainly consists of four primary parts
<table>
<thead>
<tr>
<th>Symbols</th>
<th>Name</th>
<th>Description/ functions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Process</td>
<td>Any function being performed</td>
</tr>
<tr>
<td></td>
<td>Data flow</td>
<td>Direction of data element movement</td>
</tr>
<tr>
<td></td>
<td>Data store</td>
<td>Location where data are stored</td>
</tr>
<tr>
<td></td>
<td>External entity</td>
<td>Is a source / destination of a data element</td>
</tr>
</tbody>
</table>

- Following diagram is an example for data flow diagram/ functional model
• Overview of methodologies:
  – OMT
  – Booch methodology
  – Jacobson methodology
  – Unified Approach

• UML:
  – Static and Dynamic Modelling
  – UML diagrams.
The BOOCH Methodology:-

- It is a widely used OO method that helps to design the system using object paradigm
- Covers analysis and design phase of an OO System
- Lots of symbols and diagrams are in BOOCH methodology
- But designing only less symbols and diagram are used
- It consists of six diagram for designing
1. Class diagram
2. Object diagram
3. State Transition diagram
4. Module diagram
5. Process diagram
6. Interaction diagram

- BOOCH Methodology prescribes 2 different process
  1. Macro Development Process
  2. Micro Development Process
1. Macro Development Process

- Serve as a controlling framework for the microprocessor and can take weeks or even months
- Primary concern is technical management of the system
- Consists of 5 different steps
  1. Conceptualization
  2. Analysis and development of the model
  3. Design or create the system architecture
  4. Evolution or implementation
  5. Maintenance
1. Conceptualization:-
core requirement of the system, prove the concept

2. Analysis and development of the model:-
use class diagram to describe the role and responsibility an object are to carry, interaction/behavior diagram, refine the system by many iteration and produce a stream of software implementation

3. Evolution or implementation:-
refine the system by many iteration and produce a stream of software implementation

4. Maintenance:-
add new requirement and remove bugs
2. Micro Development Process

- Each macro development process have its own micro development process
- It is the description of day to day description
- It consists of following steps
  1. Identify classes and objects
  2. Identify classes and objects semantics
  3. Identify classes and objects relationships
  4. Identify classes and objects interface and implementation
• Overview of methodologies:
  – OMT
  – Booch methodology
  – Jacobson methodology
  – Unified Approach

• UML:
  – Static and Dynamic Modelling
  – UML diagrams.
Jacobson methodology

- It covers the entire life cycle and stress traceability between the different phases, both forward and backward, eg., OOSE, OOBE, Objectory (object factory for s/w development)
- Enables **reuse of analysis and design**, reduce development time than reuse of code
- At the heart of their methodologies is the **usecase** concept, which evolved **objectory**.
1. Use cases

- are scenarios for understanding **system** requirements
- is an **interaction** between **users** & the **system**
- capture the **goal** of the user & the **responsibility** of the system to its users.
- in the requirements analysis, the use cases are described as one of the following,
  - Nonformal text with no clear flow of events
  - Text, easy to read but with a clear flow of event to follow
  - Formal style using pseudo code.
Use case description must contain,

- How & when the use case begins and ends
- Interaction between the use-case & its actors, including when the interaction occurs & what is exchanged
- How & when the use-case will need data stored in the system or will store data in the system
- Exceptions to the flow of events
- How & when the concepts of problem domain are handled

Every single use-case should have one main flow of events.

Use-case would be viewed as concrete or abstract. (is not complete and has no actors that initiate it but is used by another user case)
2. Object Oriented Software Engineering: objectory (OOSE)

- Also called objectory is a method of OO development with the specific aim to fit the development of large, real-time systems.
- Development process, called use-case driver development, stresses that use-cases are involved in several phases of the development, including analysis, design, and validation & testing.
-OOSE- is a disciplinary process for the industrialized development of s/w, based on a use-case driven design.
- Jacobson objectory has been developed & applied to numerous application areas & embedded in the CASE tools system.
Objectory is built around several different models.

1. **Use case model**: defines outside (actor) & inside (use case) of the systems behaviour.
2. **Domain object model**: object of real world are mapped into the domain obj model.
3. **Analysis object model**: it presents how the source code should be carried out & written.
4. **Implementation model**: rep implementation of the system.
5. **Test model**: constitutes the test plans, specification & reports.

Maintenance of each model is specified in its associated process.
3. Object oriented business engg. (OOBE):

- OOBE is obj modelling at the enterprise level.
  - analysis phase
    - Object Model
    - Requirement Analysis
  - design & implementation phase
    - DBMS
    - Distribution of Process
  - testing phase
    - Unit Testing
    - Integration testing
    - System Testing
Patterns & Frameworks
Patterns
Patterns:

- Each pattern describes a problem which occurs over & over again in real time, & then describes the care of the solution of that problem.
- Patterns are used to develop reusable OO s/w.
- Patterns make easier to sense successful designs & architecture.
- Patterns are invented from the proven successful solution to recurring phenomena.

Ex. In real time student, house address, mail-in etc. Are unique and identified by assigning object identifier. This recurring phenomenon is recognized as object identifier pattern (OID).
Patterns identify the key aspects of a common structure & make useful for creating reusable OO s/w.

Good patterns should have the features as,

- It solves a problem.
- It is a proven concept with a perfect track record.
- It describes a relationship of modules, system structure & mechanism in depth.
- It must have the quality of aesthetic & utility.

Patterns are generative & non-generative.
1. Generative patterns:

- Are abstract representation of system architecture and dynamic.
- Show the characteristic of a good system as well as teach how to build them.

2. Non-generative patterns:

- Static & passive.
- They just describe the recurring phenomena without telling how to reproduce them.
Pattern template:

- Every pattern must be expressed "in the form of a rule[template]" which establishes a relationship between a context, a system of forces which arises in that context & a configuration, which allows these forces to resolve themselves in that context.

- Partition should contain essential components.

- Following essential components should be clearly recognize on reading a pattern:
  
a) Name  
b) Problem  
c) Context  
d) Forces  
e) Solution  
f) examples  
g) Resulting context  
h) Rationale  
i) Related patterns  
j) Known users
Good patterns must begin normally with an abstract, which provides a short summary or general outlook.

Anti patterns

Pattern represents a “best practice”, wherever an anti pattern represents “worst practice” or “lesson learners”

Anti patterns come in two varieties:
- Those describing a bad solution to a problem that resulted in a bad situation
- Those describing how to get out of a bad situation & how to proceed from these to a good solution.
Capturing patterns:

- Process of working for patterns to document is called pattern mining (reverse architecture).
- It is important to remember that a solution in which no forces are present is not a pattern.

Guidelines:

- Focus on practicability
- Aggressive disregard of originality
- Non-anonymous review
- Writers workshops instead of presentation
- Careful editing
Frameworks
Frame works:-

- It is a set of classes providing a general solution that can be refined to provide an application or system.
- They are reusable partial application that can be specialized further to produce custom application.
- They emphasize extensibility & reuse.
- They dictate the architecture of application.
- Are a way of delivering application development patterns to support best practice shaving during application development- not just within one company, but across many companies.
Framework is a way of presenting a generic solution to a problem that can be applied to all levels in a development.

- OO s/w framework is a set of cooperative classes, reuse the class.

- Framework & patterns are related to one another, but they have different meaning.

- Framework is executable software.

- Design patterns- exp knowledge & experience about s/w.
Some difference

- Design patterns are more abstract than frameworks.
- Design patterns are smaller architectural elements than framework.
- Design patterns are less specialized than frameworks.
• Overview of methodologies:
  – OMT
  – Booch methodology
  – Jacobson methodology
  – Unified Approach

• UML:
  – Static and Dynamic Modelling
  – UML diagrams.
Unified approach:-

- A methodology for s/w development, based on methodologies by Booch, Ram, Jae & others that tries to combine the best practices, processes & guidelines along with the object management group’s UML for a better understanding of OO concept & s/w system development.
- Core concept is based on jacobson’s use case model.
- UA to s/w development revolves around the following processes & concepts
The processes are:

- Use-case driven development
- Object oriented analysis
- Object oriented design
- Instrumental development & prototyping
- Continuous testing
• Methods & technology employed include

✓ UML used for modelling
✓ Layered approach
✓ Repository for OO system development patterns & frameworks
✓ CBD
1. OO analysis:

- Identify the actors
- Develop a simple business process model using UML activity diagram
- Develop the use case
- Identify classes
2.00 design:

- Designing classes, attributes, method, association, structure, protocols
- Design access layer
- Design & prototype user interface
- User satisfaction & test based on use cases
- Iterate & define the design
3. Iterative development & continuous testing:
   - Iterative develop till the system satisfaction
   - Continuous testing in each phase

4. Modelling based on UML
   - Used for modelling
   - Std rotation for OO modelling system
   - UA uses UML to describe & model the analysis
   - Design phases of system development

5. UA proposed repository:
   - Repository - storage of code, pattern, document etc.
6. Layered approach to s/w development:

Most system developed with today’s case tools or client server application development environment tend to lean toward what is known as two-layered architecture interface & data.
A better approach to system architecture is one that isolates the function of the interface from then fun of the business.

Using three layered approach, we are able to create object that represent tangible element of your business, yet are completely independent of how they are rep to the user.
The three-layered approach consists a view or user interface layer, business layer & an access layer.
A. Business layer:

- It contains all the objects that represent the business (both data & behaviour).
- Responsibilities of business layer – to model the objects of the business & how they interact to accomplish the business process.
- When creating the business layer, it is important to keep in mind a couple things.
- These objects should not be responsible for:
  - Displaying details
  - Data access details
B. User interface (view) layer:

- This layer consists of objects with which the user interface as well as the objects needed to manage or control the interface.
- Also called view layer.
- This layer typically responsible for two major aspects.
  - Responding to user interaction.
  - Displaying business objects.
- UI layer object are identified during the OO design phase.
C. Access layer:

- It contain object that know how to communicate with the place where the data actually resides, whether it be a relational database, mainframe, internet or file.
- Regardless of where the data actually reside, the access layer has two major responsibilities.
  - Translate request
  - Translate results
- Access object are identified during OOD.
• Overview of methodologies:
  – OMT
  – Booch methodology
  – Jacobson methodology
  – Unified Approach

• UML:
  – Static and Dynamic Modelling
  – UML diagrams.
Introduction:

Model:-

→ Is an abstract representation of a system, constructed to understand the system (Is any process or structure) prior to building /modifying it

→ Modelling enables us to cope with the complexity of a system.

→ Most modelling technique used for analysis and design involve graphic languages (set of symbols (are used according to certain rules of methodology for communicating the complex relationships of information))
Main goal of CASE tool is using these graphical language along with association and methodologies.

Modelling is done in most of the phases of software life cycle such as analysis, design, and implementation.

Example for different models:
1. Use-Case Model
2. Domain – Object Model
3. Analysis Object Model
4. Implementation Model
5. Test Model

Models are represented – static and dynamic situations.
• Overview of methodologies:
  – OMT
  – Booch methodology
  – Jacobson methodology
  – Unified Approach

• UML:
  – Static and Dynamic Modelling
  – UML diagrams.
Static Model:-

- It can be viewed as a snapshot of a system’s parameters at rest or at a specific point in time.
- Are needed to represent the structural or static aspect of a system.
- Static models assume stability and an absence of change in data overtime.
- eg., Class diagram: a customer can have more than one account.
Dynamic Model:-

→ Contrast to a static model, can be viewed as a collection of procedures or behaviour that, taken together, reflects the behaviour of a system over time.
→ Dynamic relationships show how the business object interact to perform tasks.
→ System can be described by first developing its static model, which is the structure of its objects and their relationships to each other frozen in time, a baseline.
→ Useful during design and implementation phases of the system development.
→ Eg: UML Interaction Diagram and Activity Diagram, Order interact with to determine product availability.
Why Modelling?

→ Building a model for software system is like to have a blueprint for a building large construction

→ Modelling language must include

1. Model elements – fundamental modelling concepts and semantics
2. Notations – visual rendering of modelling elements
3. Guidelines – expression of usage within the trade
Use of model, a problem provides us several benefits relating to

- Clarity
- Familiarity
- Maintenance
- Simplification

Advantages

1. Designed to use complex ideas
2. Reduce complexity
3. Enhance learning and training
4. Cost is less compared to real system experience
5. Manipulation is easier
Summarize

1. It is rarely correct in the first try
2. Seeks advice and criticism of others
3. Avoid excess modelling revisions
Introduction to Unified Modelling Language:-

- UML is a language for
  - Specifying
  - Constructing
  - Visualizing
  - Documenting the s/w sym and its concepts

- Is a graphical language with a set of rules and semantics in the form of OCL (Object Constraint Language)

- Is not intended to be a visual programming language, but have a high mapping to OOP Language
Primary goals in the design of the UML were as follows:

1. Provides users a ready-to-use, expressive visual modelling language so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concept.
3. Be independent of particular programming language and development processes.
4. Provide the formal basis for understanding the modelling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts.
7. Integrated best practices and methodologies.
• Overview of methodologies:
  – OMT
  – Booch methodology
  – Jacobson methodology
  – Unified Approach

• UML:
  – Static and Dynamic Modelling
  – UML diagrams.
What is the UML?

- Unified Modeling Language
- It is a modeling language, not a process
- Rumbaugh joined Booch at Rational in 1994; in 1995, Rational added Jacobsen to their team. In 1996, work on the UML was begun.
- In January of 1997, Rational released UML 1.0 to the OMG as their proposal for a methods standard.
UML Diagrams to be Covered

1. Class Diagrams (Static)
2. Use Case Diagrams
3. Behavior Diagram (Dynamic)
   I. Interaction Diagram
      1. Collaboration Diagrams
      2. Sequence Diagrams
   II. State chart Diagram
   III. Activity Diagram
4. Implementation Diagram
   I. Component Diagrams
   II. Deployment Diagrams
Class Diagrams

• Are the most fundamental UML Diagram.
• Describe the classes in the system, and the static relationships between classes.
• Class diagrams are used during Analysis, Design and Development.
UML Class Diagram

- **Customer**
  - Rental Invoice
    - Checkout Screen
  - Rental Item
    - DVD Movie
    - VHS Movie
    - Video Game
UML Class Diagram

- **Class**: Customer
- **Abstract Class**: Rental Item
- **Generalization**: DVD Movie, VHS Movie, Video Game
- **Simple Association**: Checkout Screen
- **Simple Aggregation**: Rental Invoice
- **Composition (Dependency)**: 1..*
- **Multiplicity**: 0..1

**Notations**:
- Triangle: Generalization
- Diamond: Composition (Dependency)
- Simple Association
- Simple Aggregation
- Multiplicity

**Date**: 12/26/2012
Parts of a Class

• Classes can have four parts
  – Name
  – Attributes
  – Operations
  – Responsibilities
• Classes can show visibility and types.
• All parts but the Name are optional.

<table>
<thead>
<tr>
<th>MyClassName</th>
</tr>
</thead>
<tbody>
<tr>
<td>+SomePublicAttribute : SomeType</td>
</tr>
<tr>
<td>-SomePrivateAttribute : SomeType</td>
</tr>
<tr>
<td>#SomeProtectedAttribute : SomeType</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ClassMethodOne()</td>
</tr>
<tr>
<td>+ClassMethodTwo()</td>
</tr>
</tbody>
</table>

-- can optionally be described here.
Object Diagrams

- An Object is an instance of a class.
- Object names are underlined.
- Object diagrams are similar to class diagrams. Many of the same notations are used.
- Object diagrams capture instances of classes, and allow the dynamic relationships to be shown.

<table>
<thead>
<tr>
<th>ThisOne : MyClassName</th>
</tr>
</thead>
<tbody>
<tr>
<td>+SomePublicAttribute : SomeType</td>
</tr>
<tr>
<td>-SomePrivateAttribute : SomeType</td>
</tr>
<tr>
<td>#SomeProtectedAttribute : SomeType</td>
</tr>
<tr>
<td>+ClassMethodOne()</td>
</tr>
<tr>
<td>+ClassMethodTwo()</td>
</tr>
</tbody>
</table>
Class and Object Diagrams

Class Diagram

Customer
+id:integer
+name:string

Rental Item
+id:integer
+released:date

Joe: Customer
+id:1667
+name:Joe Smith

Casablanca: Movie
+id:22340
+released:1942

Object Diagram

Class Name
Association Name
Object Name
Attributes
Stereotypes, Tagged Values and Constraints

- Stereotypes are shown using `<<>>`
- Tagged Values and Constraints are shown using `{ }`
Use Cases

• Describe interactions between users and computer systems (both called actors).
• Capture user-visible functions.
• Achieve discrete measurable goals.
• May be
  – small ("Make selected text bold")
  – large ("Generate a table of contents")
• Are typically used during Analysis and Design.

12/26/2012
Use Case Diagram

Actor

Customer

In-Store Customer

Telephone Customer

Clerk

Identify Movie

Open Account

Return Movie

Review Account Status

Use Case

12/26/2012
Use Case Report

- The Use Case Report provides documentation for the Use Case.
- A Use Case is not complete without the report.
- The elements of the Use Case Report are shown on the right.

- Brief description
- Precondition
- Flow of events
  - Main flow
  - Subflows
  - Alternate flows
- Postcondition
- Special Requirements
- Enclosures
  - Diagrams
  - Pictures of the UI
Extends and Includes Relationships

Includes

Customer

Find Item by Title

Find Items

Search Movie Database

Extends

Clerk

Check In Movie

Assess Late Fees

Dependency
«includes»

«includes»

Stereotype
12/26/2012
Sequence Diagrams

- Can be “morphed” from Collaboration Diagrams.
- Describe interactions between objects arranged in time sequence
- Focus on objects and classes involved in the scenario and the sequence of messages exchanged
- Associated with use cases
- Used heavily during Analysis phase and are enhanced and refined during Design phase
Sequence Diagram - Rent Movie

1. find customer()
2. search (string)
3. enter movie()
4. search (string)
5. rent (movie)
6. add(Cust, item)
7. printInvoice()
8. generateRentalTotal()
Collaboration Diagrams

- Collaboration diagrams describe object interactions organized around the objects and their links to each other.
- Focus on exchange of messages between objects through their associations.
- Appears during Analysis phase.
- Enhanced during Design phase.
Collaboration Diagram - Rent Movie

1: enter_customer()
3: enter_movies()
7: print_invoice()
5: add(customer, movies)
8: generateRentalTotal()
2: IsValidCust(CustId)
4: GetMovieByBarcode()
Component Diagram

- «library» DB Server Interface (dbsvr.dll)
- Component
- Interface
- Dependency
- «application» Video Workstation (vstation.exe)
- Note
- «library» Application Framework (appfr.dll)
- Supplied by Microsoft

12/26/2012
Deployment Diagram

Node

Phone Clerk Terminal: Clerk Client

Check Out Terminal: Clerk Client

Communication Association

«TCP/IP»

Store Server

Server DB

Store Server App

12/26/2012
Activity Diagram

1. Identify Caller
2. Obtain Name & Address
3. Current Customer?
   - [no] Open Account?
     - [no]
     - [yes] Create Account
   - [yes]

Start State
Action State
Decision
Swimlanes and Fork/Join Points

Customer
- Identify Movie
- Place Order
- Pay
- Pickup Movie

Manager
- Place Order
- Collect Money
- Deliver Movie
- Fill Order
- Fork Point

Walking Clerk
- Join Point

12/26/2012
## UML Diagram Usage

<table>
<thead>
<tr>
<th>Development Phase</th>
<th>UML Diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>Use Cases, Class Diagrams, Activity Diagrams, Collaboration Diagrams, Sequence Diagrams</td>
</tr>
<tr>
<td>Design</td>
<td>Class Diagrams, Collaboration Diagrams, Sequence Diagrams, State Diagrams, Component Diagrams, Deployment Diagrams</td>
</tr>
<tr>
<td>Development</td>
<td>Collaboration Diagrams, Sequence Diagrams, Class Diagrams, State Diagrams, Component Diagrams, Deployment Diagrams</td>
</tr>
<tr>
<td>Implementation</td>
<td>Package Diagrams, Deployment Diagrams</td>
</tr>
</tbody>
</table>

12/26/2012
Review Questions

• What is a use case? Explain with example.
• Describe the difference between patterns and frame works.
• What is UML? What is the importance of UML?
• Why do we need to model a problem?
• What is macro development process?
• Name the four primary symbols of data flow diagrams.
• Explain macro development process.
• Name the pattern templates.
• What are the processes and concepts of unified approach to software development?
• What are the steps involved on OOA of unified approach.
• What is the use of repository in Unified Approach?
• List out the relationships of use case.
• What is a model?
• What is a qualifier of an association?
• Describe the class diagram.
Bibliography

• Object oriented system development by Ali Brahami
• Object oriented methodology by Booch
• A text book on UML by Srimathi