Chapter 4:
Object-Oriented Methodologies
Goals

• Object-Oriented Methodologies
  – The Rumbaugh et al. OMT
  – The Booch methodology
  – Jacobson's methodologies
Goals (Con’t)

- Patterns
- Frameworks
- Unified Approach (UA)
- layered Architecture
Basic Definitions

• A methodology is explained as the science of methods.
• A method is a set of procedures in which a specific goal is approached step by step.
Too Many Methodologies

• **1986**: Booch came up with the object-oriented design concept, the Booch method.

• **1987**: Sally Shlaer and Steve Mellor came up with the concept of the recursive design approach.
Too Many Methodologies (Con’t)

• **1989:** Beck and Cunningham came up with class-responsibility-collaboration (CRC) cards.

• **1990:** Wirfs-Brock, Wilkerson, and Wiener came up with responsibility-driven design.

• **1991:** Peter Coad and Ed Yourdon developed the Coad lightweight and prototype-oriented approach.
Too Many Methodologies (Con't)

- **1991**: Jim Rumbaugh led a team at the research labs of General Electric to develop the **object modeling technique (OMT)**.
- **1994**: Ivar Jacobson introduced the concept of the **use case**.
Survey of Some of the Object-Oriented Methodologies

- Many methodologies are available to choose from for system development.
- Here, we look at the methodologies developed by Rumbaugh et al., Booch, and Jacobson which are the origins of the Unified Modeling Language (UML) and the bases of the UA.
  - The Rumbaugh et al. method is well-suited for describing the object model or static structure of the system.
  - The Jacobson et al. method is good for producing user-driven analysis models.
  - The Booch method detailed object-oriented design models.
Rumbaugh et. al.’s Object Modeling Technique (OMT)

- OMT describes a method for the analysis, design, and implementation of a system using an object-oriented technique.
- Class, attributes, methods, inheritance, and association also can be expressed easily.
- The dynamic behavior of objects within a system can be described using OMT Dynamic model.
- Process description and consumer-producer relationships can be expressed using OMT’s Functional model.
OMT (Con’t)

• OMT consists of four phases, which can be performed iteratively:
  
  – 1. *Analysis*. The results are objects and dynamic and functional models.

  – 2. *System design*. The result is a structure of the basic architecture of the system.
3. **Object design.** This phase produces a design document, consisting of detailed objects and dynamic and functional models.

4. **Implementation.** This activity produces reusable, extendible, and robust code.
OMT Modeling

- OMT separates modeling into three different parts:
  - 1. An object model, presented by the object model and the data dictionary.
  - 2. A dynamic model, presented by the state diagrams and event flow diagrams.
OMT Object Model

• The object model describes the structure of objects in a system:
• Their identity, relationships to other objects, attributes, and operations
• The object model is represented graphically with an object diagram
• The object diagram contains classes interconnected by association lines
OMT Dynamic Model

- OMT dynamic model depict states, transitions, events, and actions
- OMT state transition diagram is a network of states and events
- Each state receives one or more events, at which time it makes the transition to the next state.
OMT Dynamic Model

No account has been selected

Nothing is selected

Account has been selected

Select Checking or saving account

Select Checking account

Select transaction type (withdraw, deposit, transfer)

Enter the amount

Confirmation
OMT Functional Model

• The OMT DFD shows the flow of data between different process in a business
• DFD use four primary symbols:
  • **Process** is any function being performed; For Ex, verify password or PIN in the ATM system
  • **Data flow** shows the direction of data element movement: for Ex. PIN code
  • **Data store** is a location where data are stored: for ex. Account is a data store in the ATM example
  • **External entity** is a source or destination of a data element; for ex. The ATM card Reader
OMT Functional Model

Data Store

Process

Data Flow

Card Reader

PIN Code

Client Account

Process PIN Code

External Entity
FIGURE 4-3
OMT DFD of the ATM system. The data flow lines include arrows to show the direction of data element movement. The circles represent processes. The boxes represent external entities. A data store reveals the storage of data.
The Booch Methodology

- The Booch methodology covers the *analysis and design phases* of systems development.
- Booch sometimes is criticized for his large set of symbols.
- You start with class and object diagram in the analysis phase and refine these diagrams in various steps.
The Booch Methodology (Con’t)

• The Booch method consists of the following diagrams:
  – *Class diagrams*
  – *Object diagrams*
  – *State transition diagrams*
  – *Module diagrams*
  – *Process diagrams*
  – *Interaction diagrams*
The Booch Methodology (Cont’d)

- Object Modeling using Booch Notation

---

Car

- color
- manufacturer
- cost

superclass

inherits

Ford

Mustang

Taurus

Escort
The Booch Methodology (Con’t) – an alarm class state transition diagram with Booch notation

Operator::TurnOffAlarm

- Silenced
  - SoundAlarm
  - SilenceAlarm

- Enabled
- Sounding

- Enable
- Disable

- AlarmFixed

Disabled
The Booch Methodology (Con't)

• The Booch methodology prescribes
  – A macro development process
    Serve as a controlling framework for the micro process and can take weeks or even months
    The primary concern of the macro process is technical management of the system
  – A micro development process.
The Macro Development Process

• The macro development process consists of the following steps:

1. Conceptualization:
   - you establish the core requirements of the system
   - You establish a set of goals and develop a prototype to prove the concept

2. Analysis and development of the model.

You use the class diagram to describe the roles and responsibilities objects are to carry out in performing the desired behavior of the system.

You use the Object diagram to describe the desired behavior of the system in terms of scenarios or use the interaction diagram.
The Macro Development Process

– 3. Design or create the system architecture.

In this phases, You use the class diagram to decide what class exist and how they relate to each other.

Object diagram to used to regulate how objects collaborate.

Then use module diagram to map out where each class and object should be declared.

Process diagram – determine to which processor to allocate a process.

– 4. Evolution or implementation. – refine the system through many iterations.

– 5. Maintenance. – make localized changes the system to add new requirements and eliminate bugs.
The Micro Development Process

• Each macro development process has its own micro development process

• The micro process is a description of the day-to-day activities by a single or small group of s/w developers

• The micro development process consists of the following steps:
  – 1. Identify classes and objects.
  – 2. Identify class and object semantics.
  – 3. Identify class and object relationships.
  – 4. Identify class and object interfaces and implementation.
The Jacobson et al. Methodologies

• The Jacobson et al. methodologies (e.g., OOBE, OOSE, and Objectory) cover the entire life cycle and stress traceability between the different phases.

• Both forward and backward
Use cases are scenarios for understanding system requirements.

A use case is an interaction between users and a system.

The use-case model captures the goal of the user and the responsibility of the system to its users.
Use Cases (Con't)

• The use case description must contain:
  – *How* and *when* the use case begins and ends.
  – The interaction between the use case and its actors, including *when* the interaction occurs and *what* is exchanged.
Use Cases (Con’t)

- **How** and **when** the use case will store data in the system.
- **Exceptions** to the flow of events.
  - Every single use case should describe one main flow events
  - An exceptional or additional flow of events could be added
  - The exceptional use case extends another use case to include the additional one
  - The use-case model employs **extends** and **uses** relationships
  - The **extends relationship** is used when you have one use case that is similar to another use case but does a bit more
Use Cases (Con’t)

• The *uses* relationships reuse common behavior in different use cases
• Use cases could be viewed as a *concrete* or *abstract*
• *Abstract use case* is not complete and has no actors that initiate it but is used by another use case.
Use case (cont..)

**FIGURE 6-9**
Transaction use cases.
Object-Oriented Software Engineering: Objectory

- Object-oriented software engineering (OOSE), also called Objectory, is a method of object-oriented development with the specific aim to fit the development of large, real-time systems.
Objectory (Con’t)

• Objectory is built around several different models:
  – **Use case model.** – defines the outside (actors) and inside (use case) of the system behavior
  – **Domain object model.** The object of the “real” world are mapped into the domain object model
  – **Analysis object model.** – how the source code (implementation) should be carried out and written
  – **Implementation model.** – represents the implementation of the system
  – **Test model.** – constitute the test plan, specifications, and reports
Object-Oriented Business Engineering (OOBE)

- Object-oriented business engineering (OOBE) is object modeling at the enterprise level.
- Use cases again are the central vehicle for modeling, providing traceability throughout the software engineering processes.
Use-case model

Express in
Structured by
Realized by
Implemented by
Tested in

Domain Object model
Analysis model
Design model
Implementation model
Testing model

OK
NOT OK
• OOBE consists of: object modeling at enterprises level
  – Analysis phase
  – Design
  – Implementation phases and
  – Testing phase.
Patterns

• A pattern is an useful information that captures the essential structure and insight of a successful family of proven solutions to a recurring problem that arises within a certain context and system of forces.

• Its help software developers resolve commonly encountered, difficult problems and a vocabulary for communicating insight and experience about these problems and their solutions.
Patterns (Con’t)

• The main idea behind using patterns is to provide documentation to help categorize and communicate about solutions to recurring problems.

• The pattern has a name to facilitate discussion and the information it represents.
Patterns (Con’t)

• A good pattern will do the following:

• *It solves a problem.* Patterns capture solutions, not just abstract principles or strategies.

• *It is a proven concept.* Patterns capture solutions with a track record, not theories or speculation.
Patterns (Con’t)

• **The solution is not obvious.** The best patterns generate a solution to a problem indirectly—a necessary approach for the most difficult problems of design.

• **It describes a relationship.** Patterns do not just describe modules, but describe deeper system structures and mechanisms.
AntiPatterns

• A pattern represents a “best practice” whereas an antipattern represents “worst practice” or a “lesson leaned”

• Antipattern come in two verities:
  • Those describe a bad solution to a problem that resulted in a bad situation
  • Those describing how to get out of a bad situation and how to proceed from there to a good solution
Patterns (Con’t)

• The pattern has a significant human component.

• All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility.
Capturing Patterns

- Guidelines for capturing patterns:
  - Focus on practicability.
  - Aggressive disregard of originality.
  - Nonanonymous review.
  - Writers' workshops instead of presentations.
  - Careful editing.
Frameworks

• A *framework* is a way of presenting a generic solution to a problem that can be applied to all levels in a development.

• A single framework typically encompasses several design patterns and can be viewed as the implementation of a system of design patterns.
Differences Between Design Patterns and Frameworks

• **Design patterns are more abstract than frameworks.**

• **Design patterns are smaller architectural elements than frameworks.**

• **Design patterns are less specialized than frameworks.**
The Unified Approach

- The idea behind the UA is not to introduce yet another methodology.
- The main motivation here is to combine the best practices, processes, methodologies, and guidelines along with UML notations and diagrams.
The Unified Approach (UA)

- The unified approach to software development revolves around (but is not limited to) the following processes and components.
UA Processes (Con’t)

• The processes are:
  – Use-case driven development.
  – Object-oriented analysis.
  – Object-oriented design.
  – Incremental development and prototyping.
  – Continuous testing.
**UA Methods and Technology**

- The methods and technology employed includes:
  - Unified modeling language (UML) used for modeling.
  - Layered approach.
  - Repository for object-oriented system development patterns and frameworks.
  - Promoting Component-based development.
UA Object-Oriented Analysis: Use-Case Driven

• The use-case model captures the user requirements.

• The objects found during analysis lead us to model the classes.

• The interaction between objects provide a map for the design phase to model the relationships and designing classes.
UA Object-Oriented Analysis: Use-Case Driven

• OOA Process consists of the following steps:

1. Identify the Actors
2. Develop the simple business process model using UML activity diagram
3. Develop the Use Case
4. Develop interaction diagrams
5. Identify classes
UA Object-Oriented Design

• Booch provides the most comprehensive object-oriented design method.
• However, Booch methods can be somewhat imposing to learn and especially tricky to figure out where to start.
• UA realizes this by combining Jacobson et al.'s analysis with Booch's design concept to create a comprehensive design process.
UA Object-Oriented Design

• OOD Process consists of:
• Design classes, their attributes, methods, associations, structures and protocols, apply design axioms
• Design the Access Layer
• Design and prototype User Interface
• User satisfaction and usability Test based on the usage/Use cases
Iterative Development and Continuous Testing

- The UA encourages the integration of testing plans from day 1 of the project.
- Usage scenarios or Use Cases can become test scenarios; therefore, use cases will drive the usability testing.
- You must iterate and reiterate until you are satisfied with the system.
Modeling Based on the Unified Modeling Language

- The UA uses the unified modeling language (UML) to describe and model the analysis and design phases of system development.
The UA Proposed Repository

- The requirement, analysis, design, and implementation documents should be stored in the repository, so reports can be run on them for traceability.

- This allows us to produce designs that are traceable across requirements, analysis, design, implementation, and testing.
The Layered Approach to Software Development

• Most systems developed with today's CASE tools or client-server application development environments tend to lean toward what is known as *two-layered architecture*: interface and data.
Two-Layer Architecture

• In a two-layer system, user interface screens are tied directly to the data through routines that sit directly behind the screens.
Problem With the Two-Layer Architecture

- This approach results in objects that are very specialized and cannot be reused easily in other projects.
Three-Layer Architecture

• Your objects are completely independent of how:
  – they are represented to the user (through an interface) or
  – how they are physically stored.
User Interface layer

This layer is typically responsible for two major aspects of the applications:

- Responding to user interaction
- Displaying business objects.
Business Layer

- The responsibilities of the business layer are very straightforward:
- model the objects of the business and how they interact to accomplish the business processes.
These objects **should not be responsible for:**

- Displaying details
- Data access details
Access Layer

- The access layer contains objects that know how to communicate with the place where the data actually resides,
- Whether it be a relational database, mainframe, Internet, or file.
Access Layer

- The access layer has two major responsibilities:
- Translate request
- Translate result
Three-Layered Architecture
Summary

• we looked at current trends in object-oriented methodologies, which have been toward combining the best aspects of today's most popular methods.
Summary (Con’t)

• Each method has its strengths. Rumbaugh et al. have a strong method for producing object models.
• Jacobson et al. have a strong method for producing user-driven requirement and object-oriented analysis models.
• Booch has a strong method for producing detailed object-oriented design models.
Summary (Con’t)

• Each method has weakness, too. While OMT has strong methods for modeling the problem domain, OMT models cannot fully express the requirements.

• Jacobson, although covering a fairly wide range of the life cycle, does not treat object-oriented design to the same level as Booch, who focuses almost entirely on design, not analysis.
Summary (Con’t)

• The UA is an attempt to combine the best practices, processes, and guidelines along with UML notations and diagrams for better understanding of object-oriented concepts and object-oriented system development.
Chapter 5:
Unified Modeling Language
Goals

• Modeling.

• Unified modeling language.
  – Class diagram.
  – Use case diagram.
  – Interaction diagrams.
    • Sequence diagram.
    • Collaboration diagram.
Goals (Con’t)

– Statechart diagram.
– Activity diagram.
– Implementation diagrams.
  • Component diagram.
  • Deployment diagram.
Introduction

• A model is an abstract representation of a system, constructed to understand the system prior to building or modifying it.
• Most of the modeling techniques involve graphical languages.
• These graphics languages are set of symbols.
Static or Dynamic Models

• Models can represent
  – static or
  – dynamic situations.
Static Model

- A static model can be viewed as a "snapshot" of a system's parameters at rest or at a specific point in time.

- Static models are needed to represent the structural or static aspect of a system.

- For Ex. A customer could have more than one account or an order could be aggregated from one or more line items: UML class diagram is an example of static model.
**Dynamic Model**

- Is a collection of procedures or behaviors that, taken together, reflect the behavior of a system over time.
- Dynamic relationships show how the business objects interact to perform task.
- For example, an order interacts with inventory to determine product availability.
- Dynamic modeling is most useful during the design and implementation phases of the system development.
Why Modeling?

- Building a model for a software system prior to its construction is as essential as having a blueprint for building a large building.
- Good models are essential for communication among project teams.
- Turban cites the following advantages:
  - Models make it easier to express complex ideas.
  - For example, an architect builds a model to communicate ideas more easily to clients.
Advantages of Modeling (Con't)

• Models reduce complexity by separating those aspects that are unimportant from those that are important.
Advantages of Modeling (Con’t)

• Models enhance learning.
• The cost of the modeling analysis is much lower than the cost of similar experimentation conducted with a real system.
• Manipulation of the model (changing variables) is much easier than manipulating a real system.
Modeling Key Ideas

• A model is **rarely correct on the first try.**

• **Always seek the advice and criticism of others.**

• **Avoid excess model revisions,** as they can distort the essence of your model. Let simplicity and elegance guide you through the process.
The Unified Modeling Language (UML)

- The unified modeling language (UML) is a language for specifying, constructing, visualizing, and documenting the software system and its components.
- UML is a graphical language with a set of rules and semantics.
- The rules and semantics of a model are expressed in English, in a form of known as Object constraint language (OCL).
UML Diagrams

The UML defines nine graphical diagrams:

1. Class diagram (static)
2. Use-case diagram
3. Behavior diagrams (dynamic):
   - 3.1. Interaction diagram:
     • 3.1.1. Sequence diagram
     • 3.1.2. Collaboration diagram
UML Diagrams

- 3.2. Statechart diagram
- 3.3. Activity diagram

4. Implementation diagram:
  4.1. Component diagram
  4.2. Deployment diagram
UML Class Diagram (object Modeling),

- The UML class diagram is the main static analysis diagram.
- Class diagrams show the static structure of the model.
- Class diagram is collection of static modeling elements, such as classes and their relationships, connected as a graph to each other and to their contents.
- For Ex: the things that exist (such as classes), their internal structures, and their relationships to other classes.
Class Notation

• In class notation, either or both the attributes and operation compartments may be suppressed.

<table>
<thead>
<tr>
<th>Boeing 737</th>
<th>Boeing 737</th>
</tr>
</thead>
<tbody>
<tr>
<td>length: meter</td>
<td>length: meter</td>
</tr>
<tr>
<td>fuelCapacity: Gal</td>
<td>fuelCapacity: Gal</td>
</tr>
<tr>
<td>doors: int</td>
<td>doors: int</td>
</tr>
</tbody>
</table>

lift ()
break ()
Class Interface Notation

- Class interface notation is used to describe the externally visible behavior of a class.
- For example, an operation with a public visibility.
- Identifying class interfaces is a design activity of OOSD.
- UML notation for interface is a small circle with name of the interface connected to the class.
- A class that requires the operations in the interface may be attached to the circle by a dashed arrows.

For Ex: a person object may need to interact with the BankAccount object to get the balance.
Binary Association Notation

- A binary association is drawn as a solid path connecting two classes or both ends may be connected to the same class.

```
Person

  ┌──────────────┐
  │worksFor      │
  │employer      │
  │employee      │

  └──────────────┘

Person

  ┌──────────────┐
  │marriedTo     │
  │             │
  │             │
```
Association Role

• A simple association—the technical term for it is *binary association*—is drawn as a solid line connecting two class symbols.

• The end of an association, where it connects to a class, shows the association role.
UML Association Notation

- In the UML, association is represented by an open arrow.
**Qualifier**

- A *qualifier* is an *association attribute*. For example, a person object may be associated to a Bank object.
- An attribute of this association is the **account#**.
- The **account#** is the qualifier of this association.
Multiplicity

- Multiplicity specifies the range of allowable associated classes.
- It is given for roles within associations, parts within compositions, repetitions, and other purposes.
- lower bound .. upper bound.
  - 0..1
  - 0..*
  - 1..3,7..10,15,19..*
OR Association

• An OR association indicates a situation in which only one of several potential associations may be instantiated at one time for any single object.

• This shown as a dashed line connecting two or more associations, all of which must have a class in common.

• An or association notation. A car may associate with a person or a company.
**Association Class**

- An *association class* is an association that also *has class properties*.
- An association class is shown as a class symbol attached by a dashed line to an association path.

```
Association Class

- An association class is an association that also has class properties.
- An association class is shown as a class symbol attached by a dashed line to an association path.

```

```
```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```
**N-Ary Association**

- An *n-ary association* is an association among more than two classes.
- Since *n-ary association* is more difficult to understand, it is better to convert an *n-ary association* to binary association.

![Diagram showing a class diagram with relations between Year, Class, and Student]
Aggregation

- Aggregation is a form of association.
- A **hollow diamond** is attached to the end of the path to indicate aggregation.
Composition

- Composition, also known as the *a-part-of*, is a form of aggregation with strong ownership to represent the component of a complex object.

- The UML notation for composition is a solid diamond at the end of a path.
Generalization

• Generalization is the relationship between a more general class and a more specific class.

• Generalization is displayed as directed line with a closed, hollow arrowhead at the superclass end.
Generalization cont..

• Ellipse (...) indicate that the generalization is incomplete and more sub classes exist that are not shown.
Use-Case Diagram

• The description of a use case defines **what happens in the system** when the use case is performed.

• In essence, the use-case model defines the outside (actors) and inside (use case) of the system's behavior.

• Use-case describe specific flow of events in the system.
Use-Case Diagram (Con’t)

• A use-case diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication (participation) associations between the actors and the use cases, and generalization among the use cases.
Actor Notations

• The three representations of an actor are equivalent.

<< actor >>
Customer

<< actor >>
Customer

Customer
Use-Case Diagram (Con’t)

• These relationships are shown in a use-case diagram:
  – Communication.
  – Uses.
  – Extends.
UML Dynamic Modeling (Behavior diagrams)

- Diagram we looked so far are static
- However, events happen dynamically in all systems:
  - Objects are created and destroyed,
  - **Objects send messages** to one another in an orderly fashion
  - In some system, external events trigger operations on certain objects
  - Furthermore object have states, the state of an object would be difficult to capture in static model
Behavior or Dynamic Diagrams

- Interaction diagrams:
  - Sequence diagrams
  - Collaboration diagrams
- Statechart diagrams
- Activity diagrams
UML Interaction Diagrams

• Interaction diagrams describe **how groups of objects collaborate to get the job done.**

• **Interaction diagrams capture the behavior of a single use case,** showing the pattern of interaction among objects.

• The diagram shows a number of example objects and the messages passed between those objects within the use case.
UML Sequence Diagram

- Sequence diagrams are an easy and natural way of describing the behavior of a system.
- A sequence diagram shows an interaction arranged in a time sequence.
- A Sequence diagrams has two dimensions:
  - vertical dimension – time
  - Horizontal – represent different objects
- The vertical line is called the object’s lifeline
- The lifeline represents the object’s existence during the interaction
• *However, Sequence diagrams does not show the relationships among the roles or association among the objects*
Telephone Call

Caller

Exchange

Receiver

Talk

OffHook

DialTone

Dial Number

RingTone

OffHooke

OnHook
UML Sequence Diagram con’t

- Each message is represented by an arrow between the lifelines of two objects.
- The order in which these message occur is shown top to bottom on the page.
- Each message is labeled with the message name
- The sequence diagram is an alternative way to understand the overall flow of the control of a program
UML Collaboration Diagram

- A collaboration diagram represents collaboration, which is a set of objects related in a particular context, and the exchange of their messages to achieve a desired outcome.

**Telephone Call**

**Object**

- Caller
  - 1: OffHook
  - 2: Dial Tone

- Exchange
  - 4: Ring Tone

- Receiver
  - 5: Off Hook

**Message**

- 3: Dial Number
- 6: On Hook

Talk
**UML Collaboration Diagram**

- In sequence diagram, arrows indicate the message sent within the given use case.
- In a collaboration diagram, the sequence is indicated by numbering the message.
- Numbering schemes: simplest, decimal
- Decimal: it makes it clear which operation is calling which other operation
- **Advantage:** used to examine the behavior of objects within single use case
- **Disadvantage:** they are great only for representing a single sequential process: they begin to break down when you want represent conditional looping behavior
UML Collaboration Diagram
(Con't)

Telephone Call

Object

Caller

1.1: OffHook

Exchange

2.1: Dial Tone

2.2: Ring Tone

Receiver

3.1: Off Hook

Talk

4.1: On Hook

Message

1.2: Dial Number
UML Statechart Diagram

• A statechart diagram (also called a state diagram) shows the sequence of states that an object goes through during its life in response to outside stimuli and messages.
• State is the set of values that describes an object at specific point in time
  • state is represented by state symbols
  • the transactions are represented by arrows connecting the state symbols
• Name compartment:
  • Internal transition compartment
UML Statechart Diagram

- A state diagram represents the state of the method execution (the state of the object executing the method)
- And the activities in the diagram represent the activities of the object that performs the method
- The purpose of the state diagram is to understand the algorithm involved in performing a method.
Idle

lift receiver and get dial tone

Dialing

Start
entry and start dialog
exit/stop dial tone

Dial
entry and number.append(n)

number.isValid()

digit(n)
UML Statechart Diagram

• A statechart diagram is similar to a petri net diagram, where a token (*shown by solid block dot*) represents an activity symbol.

• When the activity symbol appears within state symbol, it indicates the execution of an operation.

• An outgoing solid arrow attached to a statechart symbol indicates a transition triggered by the completion of the activity.
UML Statechart Diagram

- For Ex: Employee Object that contain name of an employee
- If the employee object receive the message \texttt{(getEmployeeName)} asking for name of the employee
- An operation contained in Employee class \texttt{(returnEmployeeName)} would be invoked
- Then it sent the message in the first place
- In this case, the state of the Employee object would not have been changed.
- Now consider the situation \texttt{(updateEmployeeAddress 2000 21^{st} Street, Seattle, Wa)} – the state of the object has been changed
Complex Transition
Two special events are *entry* and *exit*, which are reserved word can’t be used for event names.

Transition can be simple or complex.

**Simple transition** relationships between two states indicating that an *object in the first state will enter the second state* and perform certain actions when specific events occurs.

Complex transition may have multiple source and target states.

The complex transition is shown as short *heavy bar*.

The bar may have one or more solid arrows from state to the bar;

The bar may also have one or more solid arrows from *the bar to states*. 
UML Statechart Diagram

- No reason to prepare a state diagram for each class in your system.
- State diagrams are useful when you have a class that is very dynamic.
- In that situation, it often is helpful to prepare a state diagram to be sure you understand each of the possible states an object of the class could take and what event(message) would trigger each transition from one state to another.
UML Activity Diagram

- An activity diagram is a variation or special case of a state machine, in which the states are activities representing the performance of operations and the transitions are triggered by the completion of the operations.
UML Activity Diagram cont..

- State diagrams that focus on the events occurring to a single object as it respond to the messages,
- An activity diagram can be used to model an entire business process
- The purpose of an activity diagram is to provide a view of flows and what is going inside a use case or among several classes
• An activity model is similar to State diagram, where a token (block dot) represents an operation.

• An activity shown as a round box, containing the name of the operation.

• When operation symbol appears within an activity diagram or state diagram, it indicates the execution of the operation.

• An outgoing solid arrow attached to an activity symbol indicates a transition triggered by the completion of the activity.
UML Activity Diagram (Con’t)

FIGURE 5-20
An activity diagram for processing mortgage requests (Loan: Processing Mortgage Request).

- Prepare incoming documents
  - Index documents
  - Make electronic file
    - Complete request
      - Check data for life insurance
        - Calculate data for construction mortgage
          - Draw up contract mortgage-deed
            - Pay provision to insurance agent
              - Draw up insurance policy
Calculate payroll

[hours ≤ 40]

Normal payroll

[hours > 40]

Overtime, get authorization
**Swimlanes.** Represents responsibility for part of the overall activity and may be implemented by one or more objects.
Implementation diagrams

- These diagrams show the implementation phase of systems development.
- Such as the source code structure and the run-time implementation structure.
Implementation diagrams (Con't)

• There are two types of implementation diagrams:
  – Component diagrams show the structure of the code itself.
  – Deployment diagrams show the structure of the run-time system.
Model the physical components (source code, exe, UI) in a design. Components connected by dependency relationships.

A component is represented by the boxed figure shown in the above figure. Dependency is shown as a dashed arrow.
Deployment Diagram

Node 1: AdminServer
- Access
  - Update

Node 2: John’s PC
- UI

Shows the configuration of run-time processing elements and the software components, process, and objects live in the system.

Deployment diagrams to show how physical modules of code are distributed on various h/w platform.
Model Management: Package

- A package is a grouping of model elements.
- Packages themselves may contain other packages.
- A package may contain both subordinate packages and ordinary model elements.
Model Management: Package

- A package is represented as a folder, shown as a large rectangle with a tab attached to its upper left corner.
- If contents of the package are not shown, then the name of the package is placed within the large rectangle.
- If contents of the package are shown, then the name of the package may be placed on the tab.
A Package and Its Contents

GradeNoteBook

Year

* semester

* Class

* Student

GradeBook

grade
exam
lab
A Package and Its Dependencies

- Customer
- Business Model
  - Clients
  - Bank
    - Account
      - Checking
      - Saving
Model Constraints and Comments

- Constraints are assumptions or relationships among model elements specifying conditions and propositions that must be maintained as true otherwise the system described by the model would be invalid.

![Diagram of Person and Department with relationships: Person → WorkFor → Department, Person ∈ Department subset, ManagerOf relationship with cardinality 1-1]
Note

- A note is a graphic symbol containing textual information; it also could contain embedded images.

```
Person
  employee
    employer

Company

Static models & revision levels released yesterday

Represents an incorporated entity
```
Stereotype

- **Stereotypes** represent a built-in extendibility mechanism of the UML.
- User-defined extensions of the UML are enabled through the use of stereotypes and constraints.
UML Meta-Model

• A meta-model is a model of modeling elements.

• The purpose of the UML meta-model is to provide a single, common, and definitive statement of the syntax and semantics of the elements of the UML.
UML Meta-Model (Con’t)

- Relationship
  - Generalization
  - Association
    - Association Role
      - 1 ♦
The UML meta-model describing the relationship between association and generalization. Association is depicted as composition of association roles. Here, we use UML modeling elements (such as generalization and composition) to describe the model itself, hence, the term meta-model.
Summary

• A model is a simplified representation of reality.

• The unified modeling language (UML) was developed by Booch, Jacobson, and Rumbaugh and encompasses the unification of their modeling notations.
Summary (Con’t)

- UML consists of the following diagrams:
  - Class diagram.
  - Use case diagram.
  - Sequence diagram.
  - Collaboration diagram.
Summary (Con’t)

- Statechart diagram.
- Activity diagram.
- Component diagram.
- Deployment diagram.