High Level Design

- High-level design provides a view of the system at an abstract level
- It should sketch out all the application's security needs (OS security, application security, network security etc)
- Hardware
- User Interface

  Internal Interface, External Interface

Architecture

Monolithic

- ✓ A single program does everything. It displays the user interface, accesses data, processes customer orders, prints invoices
- ✓ But lacks flexibility
- ✓ Tight coupling between the pieces of the system makes fixing them later difficult.

Client/Server

Separates pieces of the system that need to use a particular function (clients) from parts of the system that provide those functions (servers)

Component-Based

- In *component-based software engineering (CBSE), you regard the system as a collection of loosely* coupled components that provide services for each other.
- It decouples the pieces of code much as a multitier architecture does, but the pieces are all contained within the same executable program, so they communicate directlv instead of across a network.

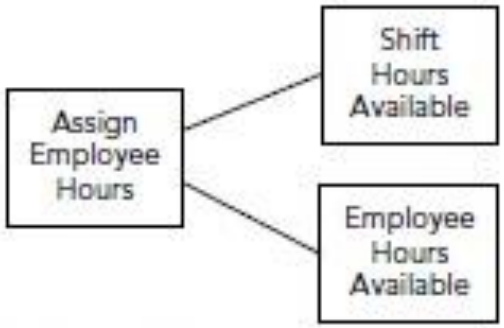

FIGURE 5-4: In a component-based architecture, components help decouple pieces of code.

Service-Oriented

- A service-oriented architecture (SOA) is similar to a component-based architecture except the pieces are implemented as services.
- A service is a self-contained program that runs on its own and provides some kind of service for its clients.
- Sometimes, services are implemented as web services

Data-Centric

- Data-centric or database-centric architectures come in a variety of flavours that all use data in some central way. The following list summarizes some typical data-centric designs:

➢ Storing data in a relational database system. This is so common that it's easy to think of as a simple technique for use in other architectures rather than an architecture of its own.

➢ Using tables instead of hard-wired code to control the application.

➢ Using stored procedures inside the database to perform calculations and implement business logic.

Event-Driven

- In an event-driven architecture (EDA), various parts of the system respond to events as they occur.
- For example, as a customer order for robot parts moves through its life cycle, different pieces of the system might respond at different times.

Rule-Based

- ✓ A rule-based architecture uses a collection of rules to decide what to do next. These systems are sometimes called expert systems or knowledge-based systems .

- Rule-based systems work well if you can identify the rules necessary to get the job done
- Rule-based systems don’t work well if the problem is poorly defined

Distributed

- In a distributed architecture, different parts of the application run on different processors and may run at the same time.
- The processors could be on different computers scattered across the network, or they could be different cores on a single computer.

- Service-oriented and multitier architectures are often distributed, with different parts of the system running on different computers.
- In general, distributed applications can be extremely confusing and hard to debug

Mix and Match

- An application doesn’t need to stick with a single architecture. Different pieces of the application might use different design approaches.
- Eg: You might create a distributed service-oriented application. Some of the larger services might use a component-based approach & Other services might use a multitier approach

Reports

- Almost any nontrivial software project can use some kinds of reports. Business applications might include reports that deal with customers

Database

- Database design is an important part of most applications.
- The first part of database design is to decide what kind of database the program will need.
- You need to specify whether the application will store data in text files, XML files, a full-fl edged relational database

Audit Trails

- An audit trail keeps track of each user who modifies (and in some applications views) a specific record.
- Auditing can be as simple as creating a history table that records a user's name, a link to the record that was modified, and the date when the change occurred

Database Maintenance

- ✓ You can move the older data into a data warehouse , a secondary database that holds older data for analysis
- ✓ Removing old data from a database can help keep it responsive
- ✓ Should design a database backup and recovery scheme

Data Flows and States

- Many applications use data that fl ows among different processes

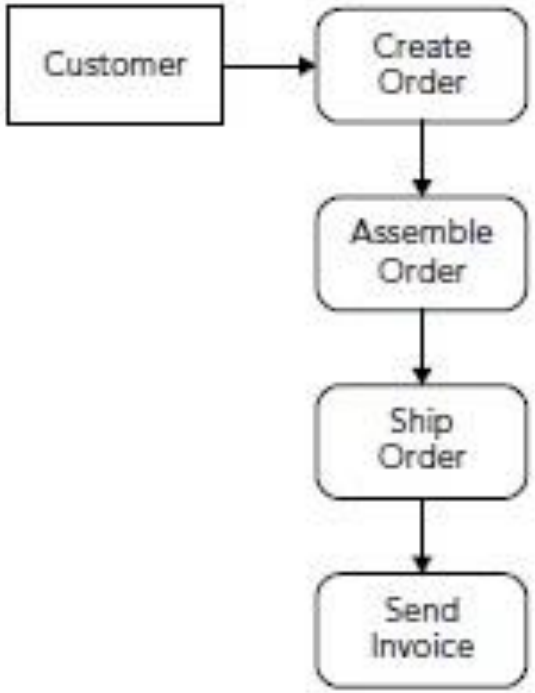

FIGURE 5-5: A data flow diagram shows how data such as a customer order flows through various processes.

- UML 2.0 defines 13 diagram types divided into three categories (and one subcategory) as shown in the following list:
- ➢ Diagram
    - ➢ Structure Diagram
        - ➢ Class Diagram
        - ➢ Composite Structure Diagram
        - ➢ Component Diagram
        - ➢ Deployment Diagram
        - ➢ Object Diagram
        - ➢ Package Diagram
        - ➢ Profile Diagram

- ➢ Behavior Diagram
  - ➢ Activity Diagram
  - ➢ Use Case Diagram
  - ➢ State Machine Diagram
  - ➢ Interaction Diagram
    - ➢ Sequence Diagram
    - ➢ Communication Diagram
    - ➢ Interaction Overview Diagram
    - ➢ Timing Diagram

Structure Diagrams

- A structure diagram describes things that will be in the system you are designing
- The following list summarizes UML's structure diagrams:

➢ Class Diagram—Describes the classes that make up the system, their properties and methods, and their relationships.

➢ Object Diagram—Focuses on a particular set of objects and their relationships at a specific time

➢ Component Diagram—Shows how components are combined to form larger parts of the system.

➢ Composite Structure Diagram—Shows a class's internal structure and the collaborations that the class allows.

- Package Diagram—Describes relationships among the packages that make up a system. For example, if one package in the system uses features provided by another package, then the diagram would show the first “importing” the second.

- Deployment Diagram—Describes the deployment of artifacts (files, scripts, executables, and the like) on nodes(hardware devices or execution environments that can execute artifacts).

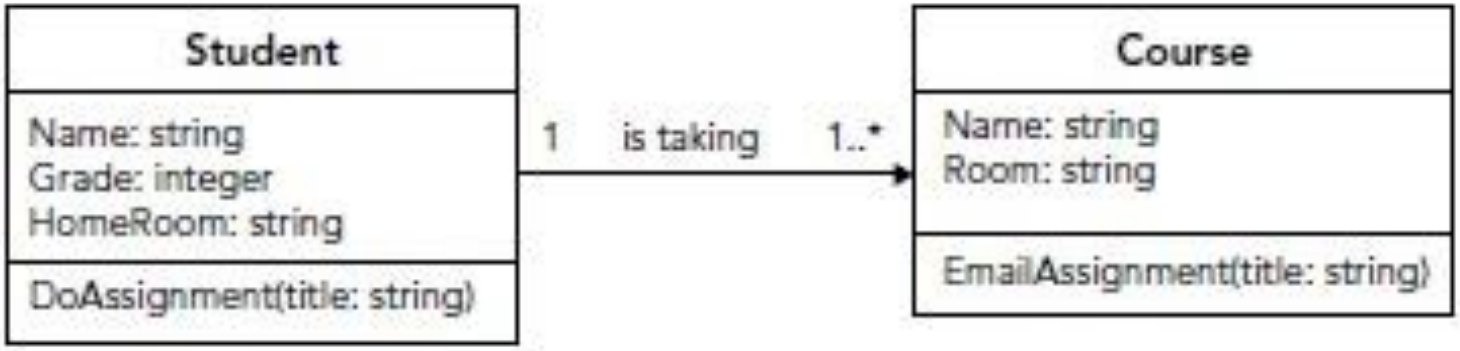

FIGURE 5-8: The relationship in this class diagram indicates that 1 Student takes 1 or more Courses.

TABLE 5-2: Class Diagram Visibility Symbols

SYMBOL	MEANING	EXPLANATION
+	Public	The member is visible to all code in the application.
–	Private	The member is visible only to code inside the class.
#	Protected	The member is visible only to code inside the class and any derived classes.
~	Package	The member is visible only to code inside the same package.

Activity Diagrams

- Represent work flow for activities

SYMBOL	REPRESENTS
Rounded rectangle	An action or task
Diamond	A decision
Thick bar	The start or end of concurrent activities
Black circle	The start
Circled black circle	The end

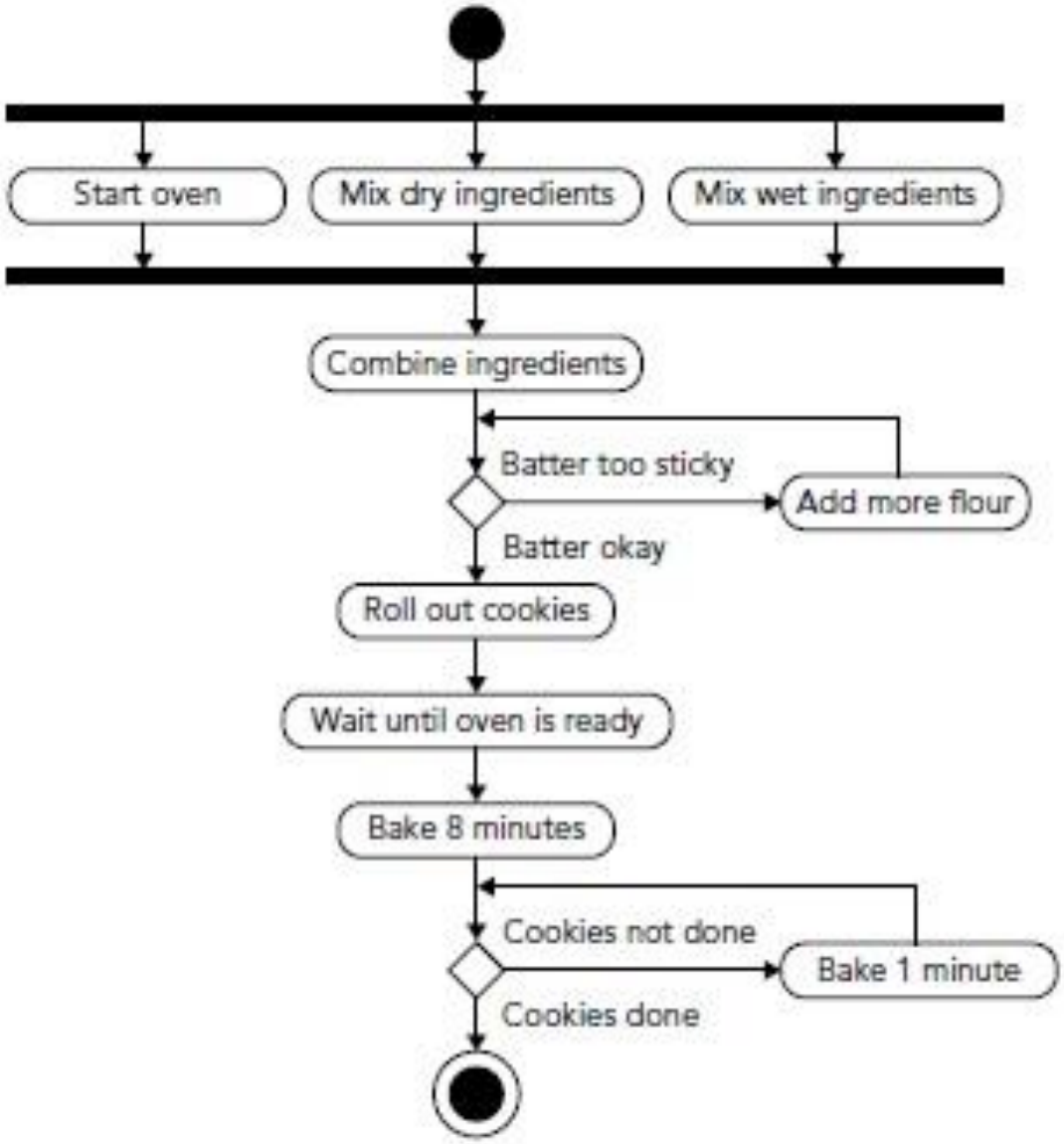

FIGURE 5-10: An activity diagram is a bit like a flowchart showing how work flows.

Use Case Diagram

- A use case diagram represents a user's interaction with the system.
- Use case diagrams show stick figures representing actors connected to tasks represented by ellipses
- To provide more detail, you can use arrows to join subtasks to tasks.

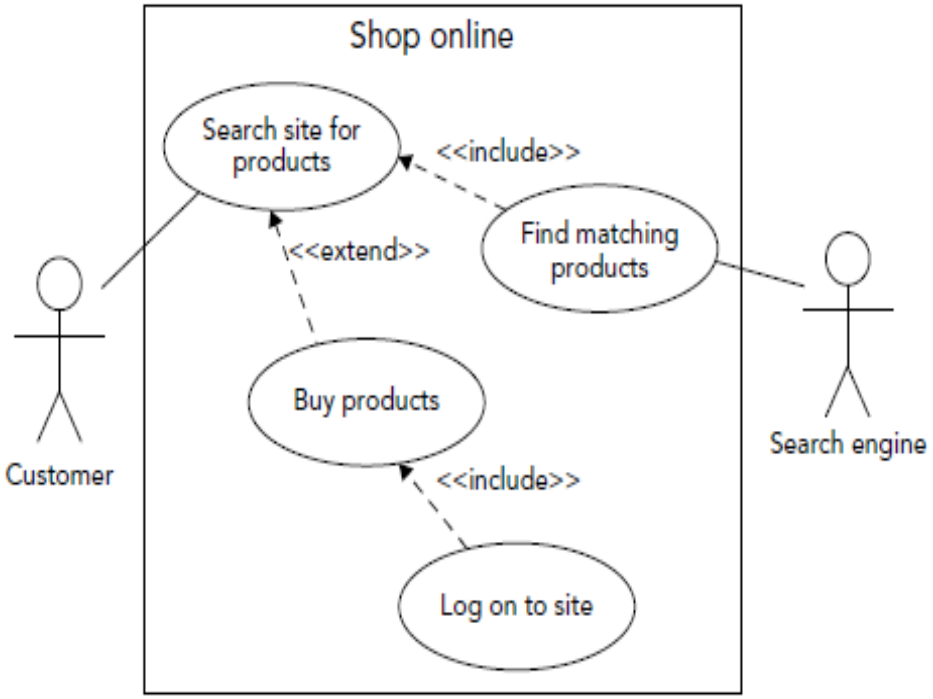

FIGURE 5-11: A use case diagram shows actors and the tasks they perform (possibly with subtasks and extensions).

State Machine Diagram

- Shows the states through which an object passes in response to various events.
- States are represented by rounded rectangles. Arrows indicate transitions from one state to another.
- Sometimes annotations on the arrows indicate what causes a transition
- Black circle represents the starting state and a circled black circle indicates the stopping state.

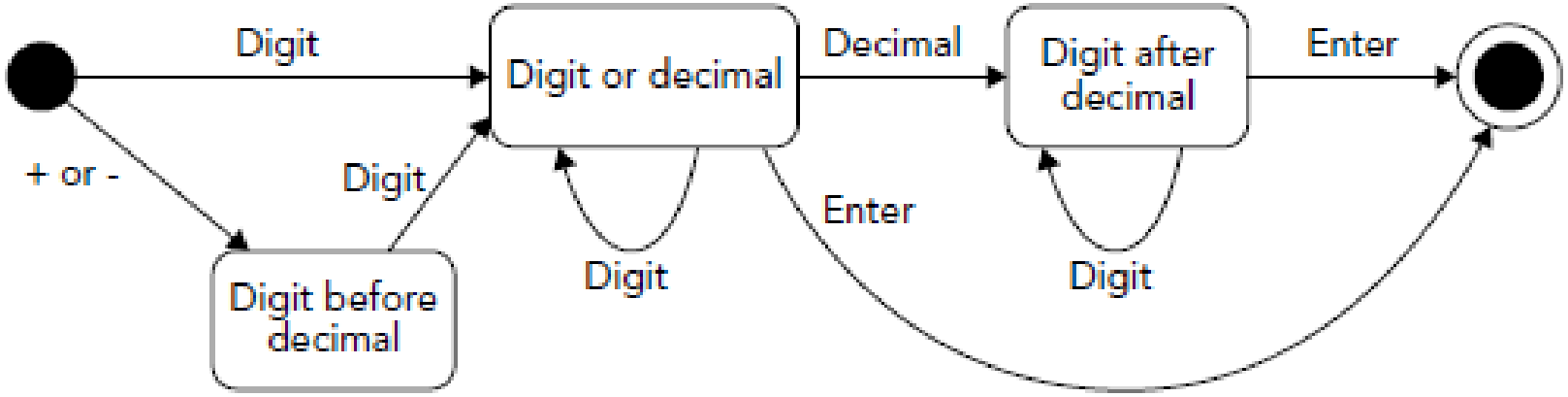

FIGURE 5-12: This state machine diagram represents reading a floating point number.

Interaction Diagrams

- Interaction diagrams are a subset of activity diagrams. They include sequence diagrams, communication diagrams, timing diagrams, and interaction overview diagrams.

Sequence Diagram

- A *sequence diagram shows how objects collaborate in a particular scenario. It represents the* collaboration as a sequence of messages.

- Objects participating in the collaboration are represented as rectangles or sometimes as stick figures for actors. They are labelled with a name or class.

- Below each of the participants is a vertical dashed line called a *lifeline*
- Labelled arrows with solid arrowheads represent synchronous messages. Arrows with open arrowheads represent asynchronous messages.

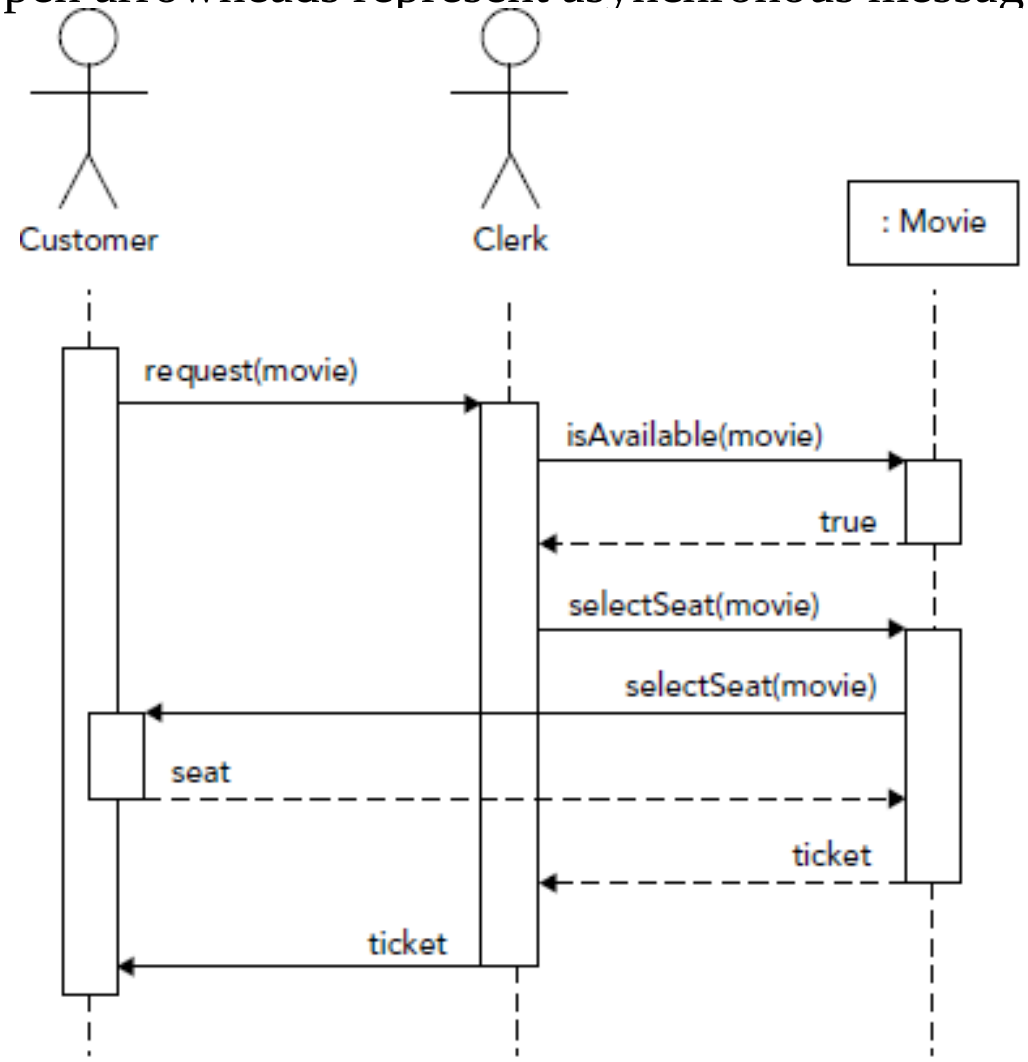

FIGURE 5-13: A sequence diagram shows the timing of messages between collaborating objects.

Communication Diagram

- *Communication diagram shows communication among objects during* some sort of collaboration
- Focuses more on the objects involved in the collaboration.
- Diagram uses lines to connect objects
- Messages are numbered that so you can follow the sequence of messages.

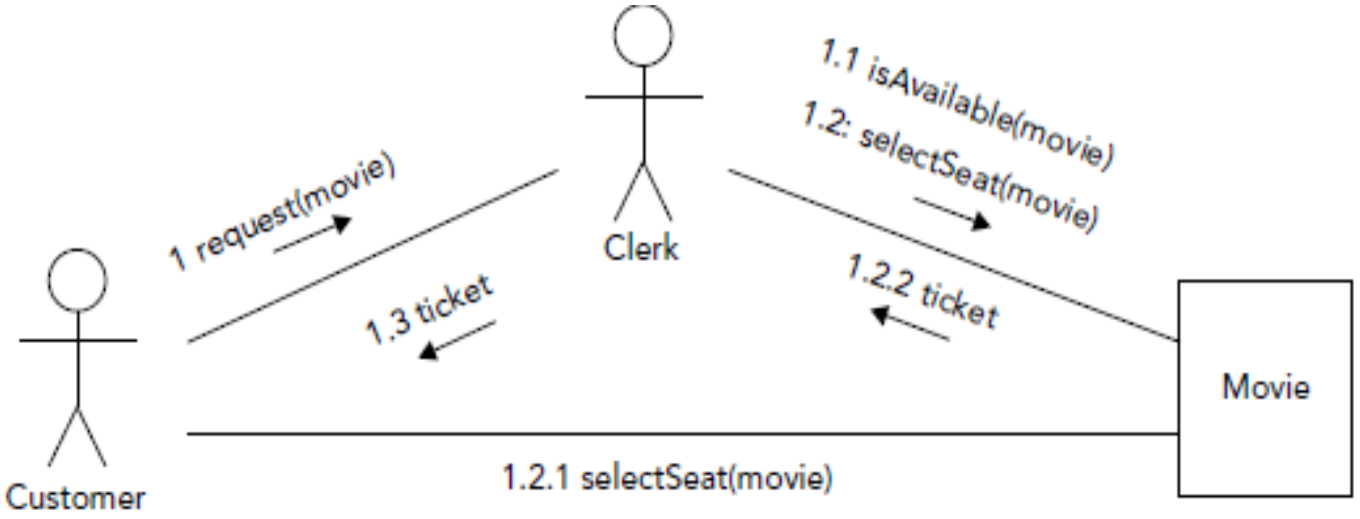

FIGURE 5-14: A communication diagram emphasizes the objects participating in a collaboration.

Timing Diagram

- A *timing diagram shows one or more objects' changes in state over time. A timing diagram looks* a lot like a sequence diagram turned sideways, so time increases from left to right.
- These diagrams can be useful for giving a sense of how long different parts of a scenario will take.

Interaction Overview Diagram

- An *interaction overview diagram is basically an activity diagram where the nodes can be frames* that contain other kinds of diagrams. Those nodes can contain sequence, communication, timing, and other interaction overview diagrams.

- This lets you show more detail for nodes that represent complicated tasks.