Unit-2

REQUIREMENTS ENGINEERING

Contents:

- Chapter 1: Requirements Understanding
- Chapter 2: Requirements Modeling

Requirements Engineering

- The most difficult tasks that face a software engineer is **understanding the requirements of a problem.**

- The broad spectrum of tasks and techniques that lead to an **understanding of requirements** is called **requirements engineering.**

- Requirements engineering builds a **bridge to design and construction.**

Requirements engineering…

- Requirements engineering **provides**
  - the appropriate mechanism for **understanding what the customer wants,**
  - **analyzing need,**
  - **assessing feasibility,**
  - **negotiating** a reasonable solution,
  - the **solution** unambiguously,
  - **validating** the specification.

Requirements engineering…

- Requirements engineering tasks lead to an understanding of :
  - What the **business impact of software** will be
  - What the **customer wants** and
  - How **end users will interact** with the software

- Software engineers or "**analysts**" and other **project stakeholders** (managers, customers, end users) **all participate** in requirements engineering.

7 distinct tasks in Requirements Engineering

1. Inception,
2. Elicitation,
3. Elaboration,
4. Negotiation,
5. Specification,
6. Validation, and
7. Management.

Requirements Engineering Task

1. **Inception**—a task that defines the **scope & nature** of the problem to be solved

✓ ask a set of questions that establish …

- Basic understanding of the problem.
- The people who want a solution.
- The nature of the solution that is desired, and
- The effectiveness of preliminary communication and collaboration between the customer and the developer.

2. **Elicitation**— a task that helps stakeholders **define what is required**.

3. **Elaboration**—create an analysis model that identifies data, function and behavioral requirements.

Requirements Engineering task…

4. **Negotiation**—agree on a deliverable system that is realistic for developers and customers

   - what are the **priorities**,
   - what is essential,
   - when is it required?

5. **Specification**— finally, the problem is specified in some manner and then **reviewed ,** can be any one (or more) of the following:
   - A written document
   - A set of models
   - A formal mathematical
   - A collection of user scenarios (use-cases)
   - A prototype

Requirements Engineering tasks…

6. **Validation**—a review mechanism that looks for
   - errors in content or interpretation
   - areas where clarification may be required
   - missing information
   - inconsistencies (a major problem when large products or systems are engineered)
   - conflicting or unrealistic (unachievable) requirements.

7. **Requirements management** is a set of activities that help the project team **identify, control, and track** requirements and changes to requirements at any time as the project proceeds

Software Requirements Specification Template

Table of Contents
Revision History

1. **Introduction**
   - 1.1 Purpose
   - 1.2 Document conventions
   - 1.3 Intended Audience and Reading Suggestions
   - 1.4 Project Scope
   - 1.5 References

2. **Overall Description**
   - 2.1 Product Perspective
   - 2.2 Product Features
   - 2.3 User Classes and Characteristics
   - 2.4 Operating Environment
   - 2.5 Design and Implementation Constraints
   - 2.6 User Documentation
   - 2.7 Assumptions and Dependencies

3. **System Features**
   - 3.1 System Feature 1
   - 3.2 System Feature 2 (and so on)

4. **External Interface Requirements**
   - 4.1 User Interfaces
   - 4.2 Hardware Interfaces
   - 4.3 Software Interfaces
   - 4.4 Communications Interfaces

5. **Other Nonfunctional Requirements**
   - 5.1 Performance Requirements
   - 5.2 Safety Requirements
   - 5.3 Security Requirements
   - 5.4 Software Quality Attributes

6. **Other Requirements**

Appendix A: Glossary
Appendix B: Analysis Models
Appendix C: Issues List

Functional Vs Non-functional Requirements

- **Functional requirements** are those which are related to the technical functionality of the system.
  - **Non-functional requirement** is a **requirement** that specifies criteria that can be used to judge the operation of a system in particular conditions, rather than specific behaviors.

- **Functional requirements** are the product features or its functions that must be designed directly for the users and their convenience. They define the functionality of the software, which the software engineers have to develop so that the users could easily perform their tasks up to the business **requirements**.
  - **Nonfunctional Requirements** (NFRs) define system attributes such as **security, reliability, performance, maintainability, scalability, and usability**.

Establishing the groundwork

- The **steps** required to establish the groundwork for an understanding of software requirements

1. Identifying Stakeholders--Stakeholder can be "**anyone who benefits** in a direct or indirect way from the system which is being developed"

Ex. Business manager, project manager, marketing people, software engineer, support engineer, end-users, internal-external customers, consultants, maintenance engineer.

- Each one of them has **different view** of the system.

2. Recognizing Multiple Viewpoints

- Many different stakeholders exist, the requirements of the system will be explored from **many different points of view.**
  - **Marketing group** concern about feature and function to excite potential market. To **sell easily** in the market.
  - **Business manager** concern about feature built **within budget** and will be ready to meet market.
  - **End user** – Easy to **learn and use**.
  - **Software Engineer** – product functioning at various infrastructure support.
  - **Support engineer** – Maintainability of software.
- Role of Requirement Engg. is to **categorize** all stakeholder information in a way that there could be no inconsistent or conflict requirement with one another.

3. Working toward Collaboration

- Requirement Engg. identify **areas of commonality** (i.e. Agreed requirement) and **areas of conflict** or **inconsistency**.

- It does not mean requirement defined by committee. It may happened they providing just view of their requirement.

- **Business manager** or **senior technologist** may make final decision.

4. Asking the Questions

- The **first set of context-free** questions focuses on the customer and other stakeholders, the overall project goals and benefits.
  - Who is behind the request for this work?
  - Who will use the solution?
  - What will be the economic benefit of a successful solution
  - Is there another source for the solution that you need?
- These questions will help – stakeholder interest in the software & measurable benefit of successful implementation.

Asking the question…

Next set of questions – **better understanding of the problem**.

- What business problem(s) will this solution address?
- Describe business environment in which the solution will be used?
- will performance or productivity issues affect the solution is approached?

Final set of questions – **Effectiveness of communication**

- Are my questions relevant to the problem?
- Am I asking too many questions?
- Can anyone else provide additional information?
- should I be asking you anything else?

Eliciting Requirements

Ms. Archana A, Asst.
Professor, PESU-MCA

Eliciting Requirements

- Requirements elicitation (also called requirements gathering) combines **elements *of* problem solving, elaboration, negotiation, and specification.**

- **Approach for eliciting requirement:**

1.Collaborative Requirement Gathering

2.Quality Function Deployment

3.User (Usage)Scenarios

4.Elicitation Work Products

1. Collaborative Requirement Gathering

- Collaborative Requirement gathering have some **basic guidelines**:
  - **Meetings** are conducted & attended by both software engineer & interested stakeholders.
  - **Rules** for preparation and participation are **established**.
  - **Agenda** should be **formal** enough to cover all important points, but informal enough to encourage the free flow of ideas.
  - A **facilitator (** customer / developer / outsider **) controls** the meeting.
  - A "**definition mechanism**" (work sheets, flip charts, wall stickers, chat room , etc.) is used.
- During the meeting:
  - The problem is identified.
  - Elements of the solution are proposed.
  - Different approaches are negotiated.
  - A preliminary set of solution requirements are obtained.
  - The atmosphere is collaborative and non-threatening.

- Flow of event – Outline the sequence of events occurs
  - Requirement gathering meeting (initial meeting)
  - During meeting
  - Follow the meeting

- In initial meeting, distribute “Product request” (defined by stakeholder) to all attendee.

- Based on product request, each attendee is asked to make
  - List of objects (Internal or external system objects)
  - List of services(Processes or functions)
  - List of constraints (cost, size, business rules) and performance criteria(speed, accuracy) are developed.

- **Collect** lists from everyone and **combine.**
- Combined list eliminates redundant entries, add new ideas , but does not delete anything.
- Objective is to develop a consensus (process of decision making) list in each topic area (objects, services, constraints and performance).
- **Based on lists, team is divided into smaller sub-teams :** each works to develop mini-specification for one or more entries on each of the lists.

- Each sub-team presents its mini-specification to all attendees for discussion.
  - Addition, deletion and further elaboration are made.
- Now each team makes a list of validation criteria for the product and present to team.
- Finally, one or more participants is assigned the task of writing a **complete draft specification**.

2. Quality Function Deployment(QFD)

- It is a technique that **translate the needs of the customer** into **technical requirement** for software.
- Concentrates on **maximizing customer satisfaction**.
- QFD emphasizes – what is valuable to the customer and then deploys these values throughout the engineering process.

QFD…

- **QFD** identifies **Three** types of requirement:

1. **Normal Requirements** – Reflect objectives and goals stated for product. If requirement are present in final products, customer is satisfied.
   - Ex: graphical displays, specific system functions etc.
2. **Expected Requirements** – Customer **does not explicitly** state them. Customer assumes it is implicitly available with the system.
   - Eg: ease of human/machine interaction, ease of software installation , etc..

3. **Exciting Requirements-** **Features** that go **beyond the customer's expectation.**

- Ex--software for a new mobile phone comes with standard features, but is coupled with a set of unexpected capabilities (e.g., multitouch screen, visual voice mail, screen recording etc.) **that delight** every user of the product.

3. User Scenarios

- A **scenario** is a **scene** that illustrates **some interaction** with a proposed system.
- Developers and users create a set of usage threads for the system to be constructed.
- A use-case scenario is a story about how someone or something **external** to the software (known as an **actor**) **interacts with the system**.
- Describe **how the system will be used.**
- Each scenario is described from the point-of-view of an "actor"—a person or device that interacts with the software in some way.

4. Elicitation Work Products

- Elicitation work product will vary depending upon the size of the system or product to be built.

- For most systems, the work products include

  - Statement of need and feasibility.
  - Statement of scope.
  - List of participants in requirements elicitation.
  - Description of the system's technical environment.
  - List of requirements and associated domain constraints.
  - List of usage scenarios.
  - Any prototypes developed to refine requirements.

Developing Use cases

Ms. Archana A, Asst.
Professor, PESU-MCA

Developing Use Cases

- A use case is a **methodology** used in system **analysis** to **identify, clarify, and organize** system requirements.
- The use case is made up of a **set of possible sequences** of ***interactions between systems and users*** in a particular environment and related to a particular goal.
- Use case tells a **stylized story** about how an end user (playing one of a number of possible roles) interacts with the system under a specific set of circumstances.

Developing use cases…

- The story may be **narrative text, an outline of tasks or interactions, a template-based description, or a diagrammatic representation**.

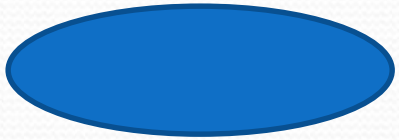
- Regardless of its form, a use case **depicts the software** or system from the **end user's point of view.**

Developing Use Cases…

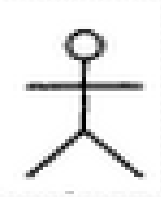
- Step One – **Define the set of actors** that will be involved in the story
  - **Actors** are **people, devices,** or **other systems** that use the system or product **within the context of the function** and behavior that is to be described
  - Actors are those that **communicate with the system** or product and that are external to the system itself
- Step Two – **Develop use cases**, where each one answers a set of questions

UML:-Use Case diagram elements.

- Use case:



- Actor:



- Association:

- An use case involves a sequence of messages among the system and its actors. This can be repeated several times.
- Consider for example **A Vending machine**
  - In a vending machine, the customer first inserts a coin and the vending machine displays the amount deposited, depending on the money inserted and the item selected the machine may or may not return change.
  - If a customer selects a beverage whose supply is exhausted, the vending machine displays a warning menu.

For Vending machine

- Actors:-Customer, Repair technician, Stock Clerk

1. A customer actor can buy a beverage from a vending machine.
2. A repair technician can perform scheduled maintenance on a vending machine
3. A stock clerk will load the items into the vending machine

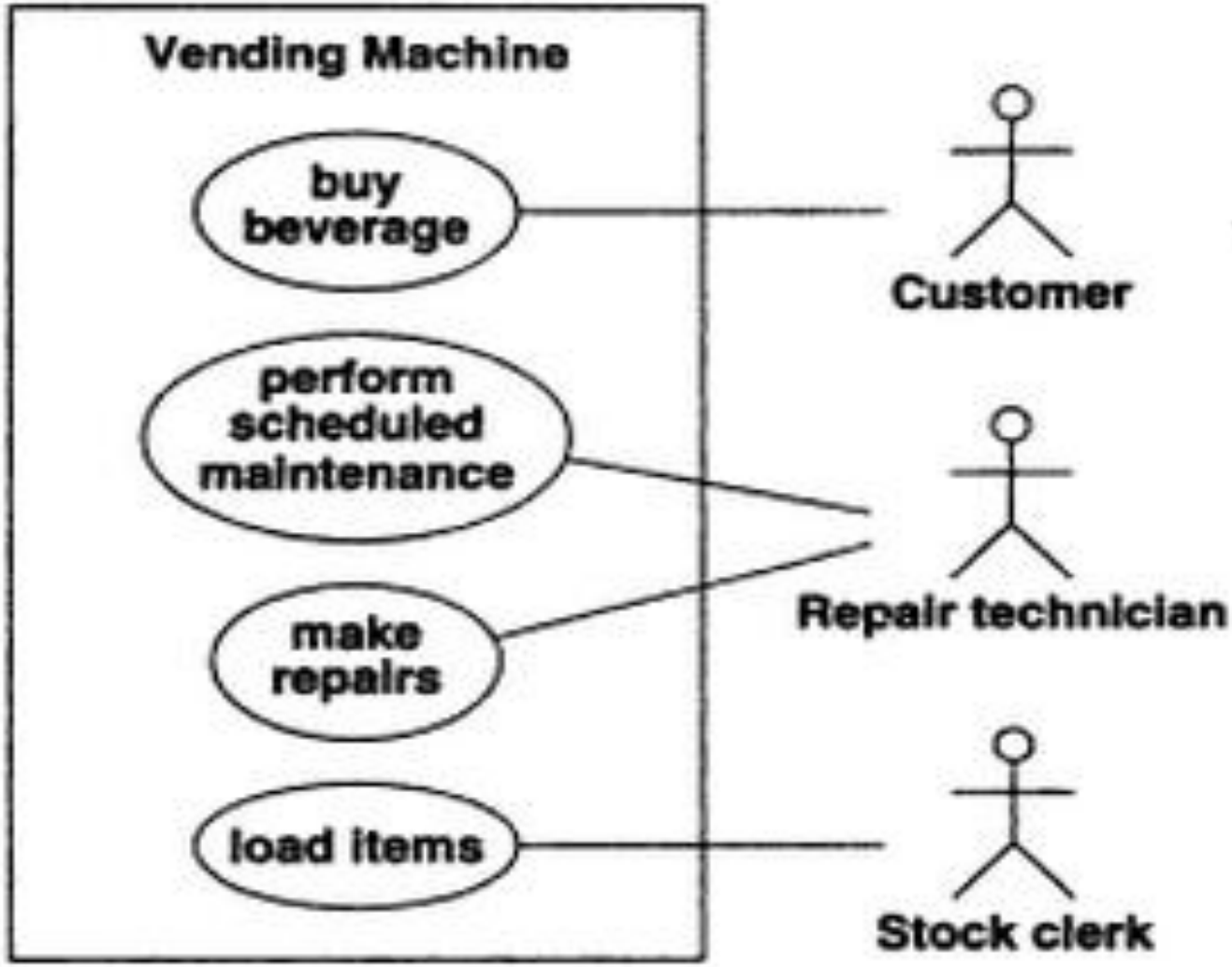


Figure 7.3 **Use case diagram for a vending machine.** A system involves a set of use cases and a set of actors.

Building the Requirements Model

Ms. Archana A, Asst. Professor, PESU-MCA

Elaboration:

Building the requirements model

- The intent of the **analysis model** is to provide a **description of the required informational, functional, and behavioral domains** for a computer-based system.

- **Elements** of the Requirements (Analysis) Model

1. Scenario-based elements
2. Class-based elements
3. Behavioral elements
4. Flow-oriented elements

Elements of the Analysis Model

- **Scenario-based elements**
  - Describe the system from the **user's point of view** using **scenarios** that are depicted in **use cases** and **sequence diagrams.**

- **Class-based elements**
  - Identify the **domain classes** for the objects manipulated by the actors, the **attributes** of these classes, and how they interact with one another; they utilize **class diagrams** to do this.

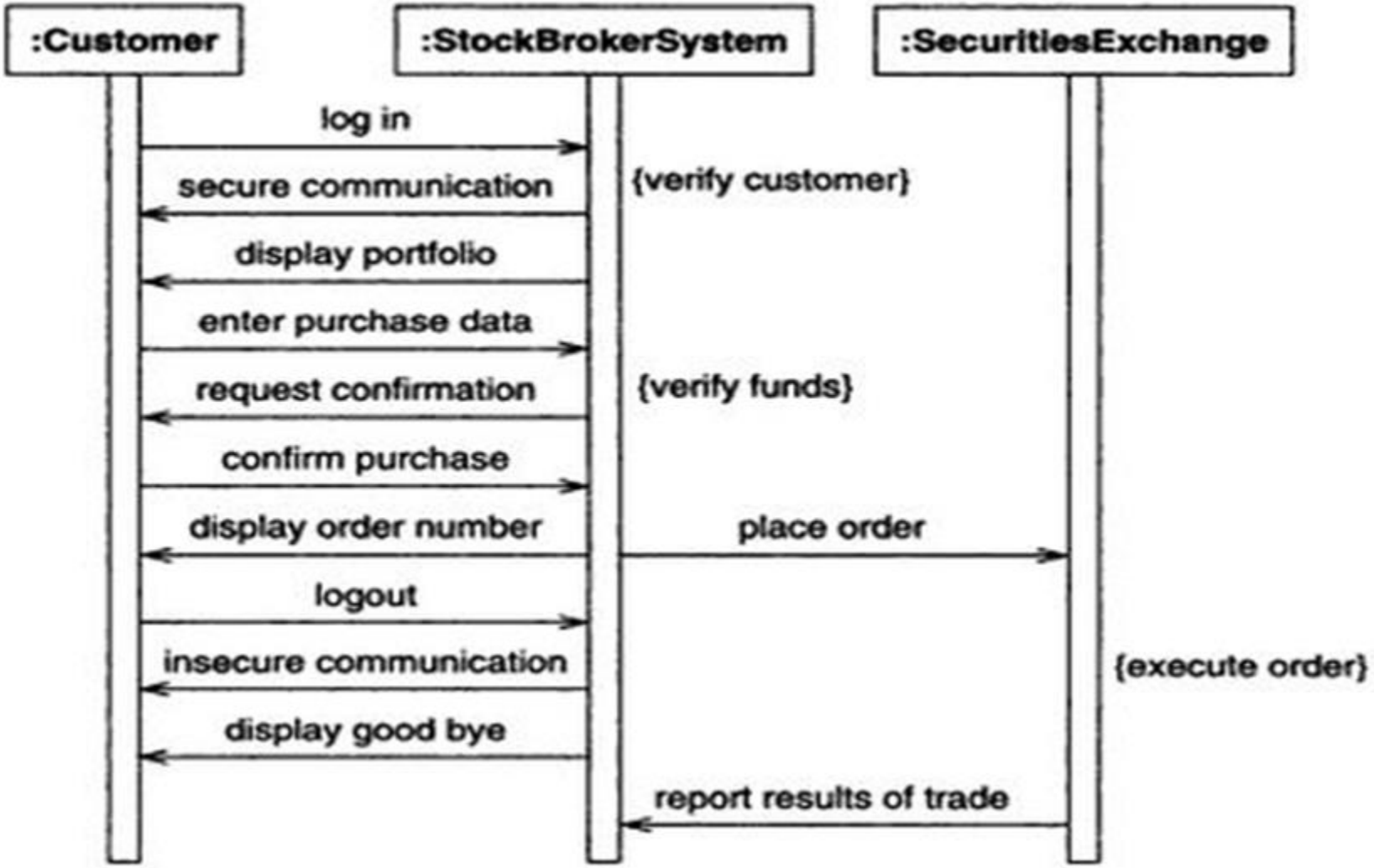
Elements of the Analysis Model…

- **Behavioral elements**
  - Use **state diagrams** to represent the state of the system, the events that cause the **system to change state**, and the actions that are taken as a result of a particular event; can also be applied to each class in the system
- **Flow-oriented elements**
  - Use **data flow diagrams** to show the **input data** that comes into a system, what **functions** are applied to that data to do transformations, and what resulting **output data** are produced

Scenario based elements- Scenario for a session with online Stock Broker

John Doe logs in.
System establishes secure communications.
System displays portfolio information.
John Doe enters a buy order for 100 shares of GE at the market price.
System verifies sufficient funds for purchase.
System displays confirmation screen with estimated cost.
John Doe confirms purchase.
System places order on securities exchange.
System displays transaction tracking number.
John Doe logs out.
System establishes insecure communication.
System displays good-bye screen.
Securities exchange reports results of trade.

Eg:Sequence diagram- Scenario based elements

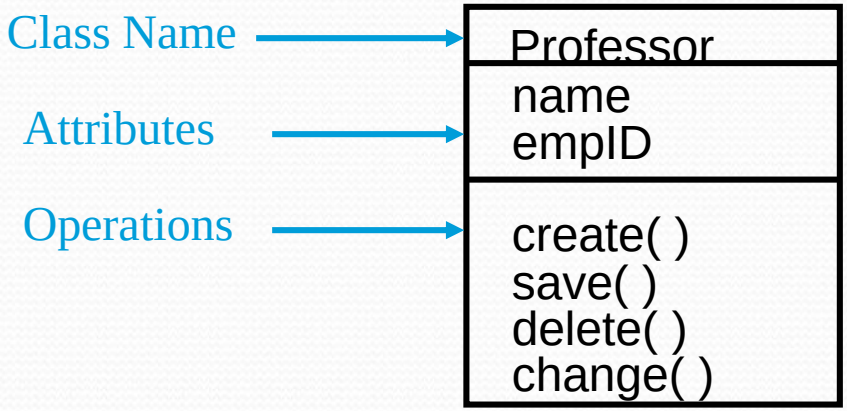


class diagram

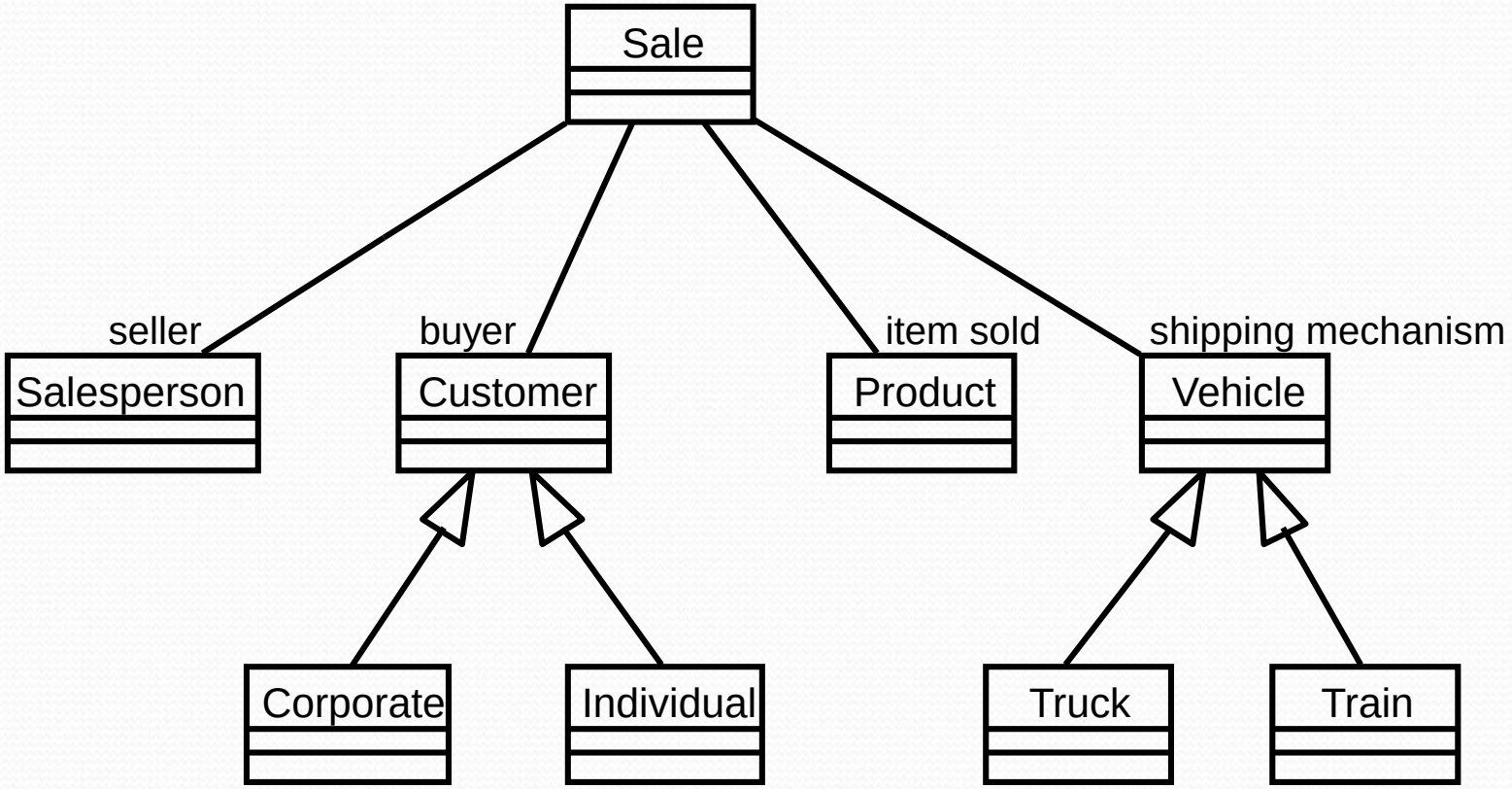
- The class diagram is a **static diagram**. It represents the **static view** of an application.
- The class diagram describes the **attributes and operations** of a class and also the **constraints** imposed on the system.
- The class diagram shows a **collection of classes, interfaces, associations, collaborations and constraints**. It is also known as a ***structural diagram***.
- Purpose--**Analysis and design** of the static view of an application.

Class Compartments

- A class is comprised of three sections
  - The first section contains the class name
  - The second section shows the structure (attributes)
  - The third section shows the behavior (operations)

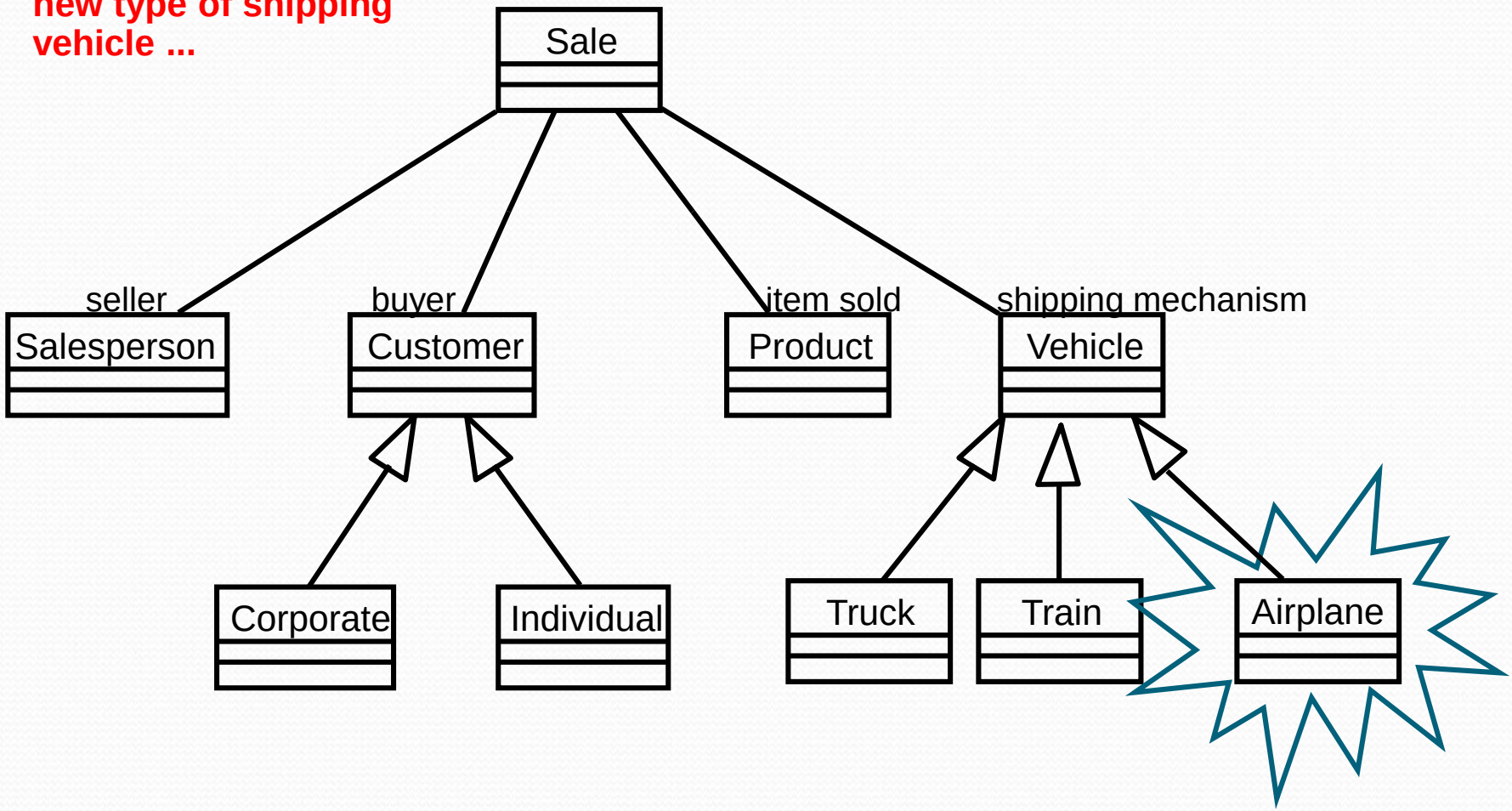


Class Diagram for the Sales - Example for class based elements



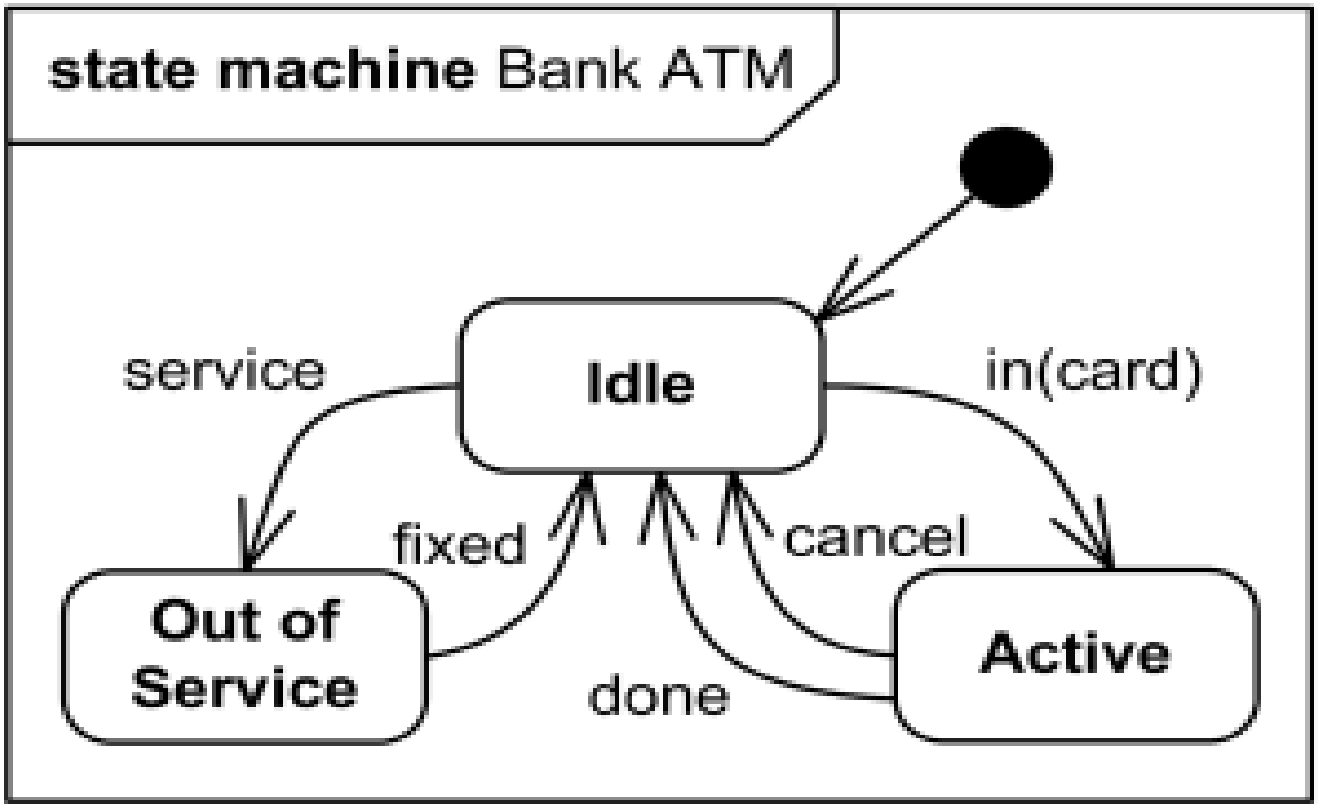
Effect of Requirements Change

Suppose you need a new type of shipping vehicle ...



Change involves adding a new subclass

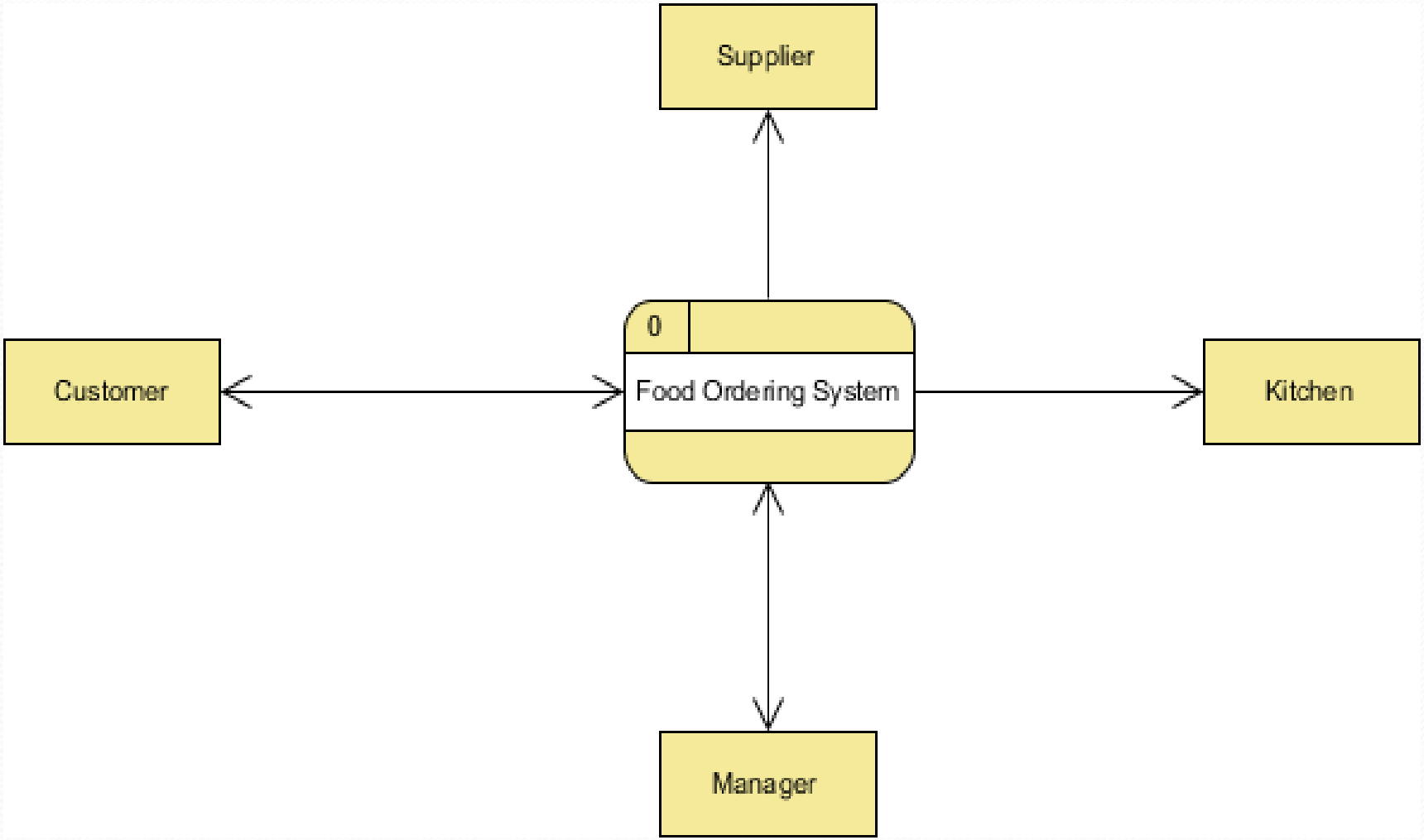
State diagram – Behavioral elements



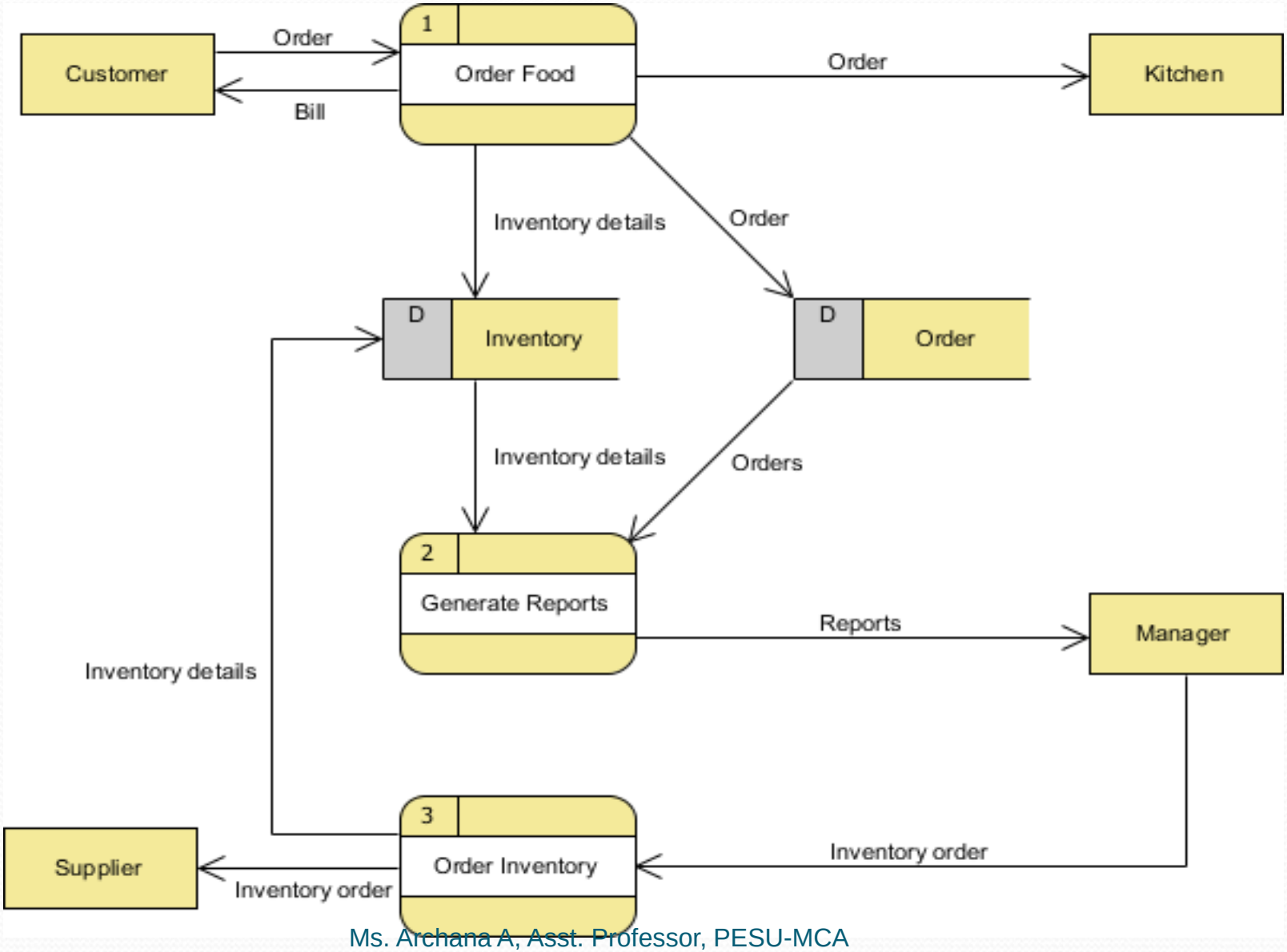
Data Flow Diagram (DFD)

- Usually begin with drawing a **context diagram**, a simple representation of the whole system.
- To **elaborate further** from that, drill down to a **level 1 diagram** with additional information about the **major functions** of the system.
- This could continue to evolve to become a **level 2** diagram when further analysis is required.
- Progression to level 3, 4 and so on is possible .

The Food Ordering System –Context DFD



Level-1 DFD


Customer
Order
Bill
1
Order Food
Order
Kitchen
Inventory details
Order
D
Inventory
D
Order
Inventory details
Orders
2
Generate Reports
Reports
Manager
Inventory details
Supplier
Inventory order
3
Order Inventory
Inventory order

Requirements Modeling

Requirements Modeling Strategies

- **There are two approaches to requirements analysis model. They are:**

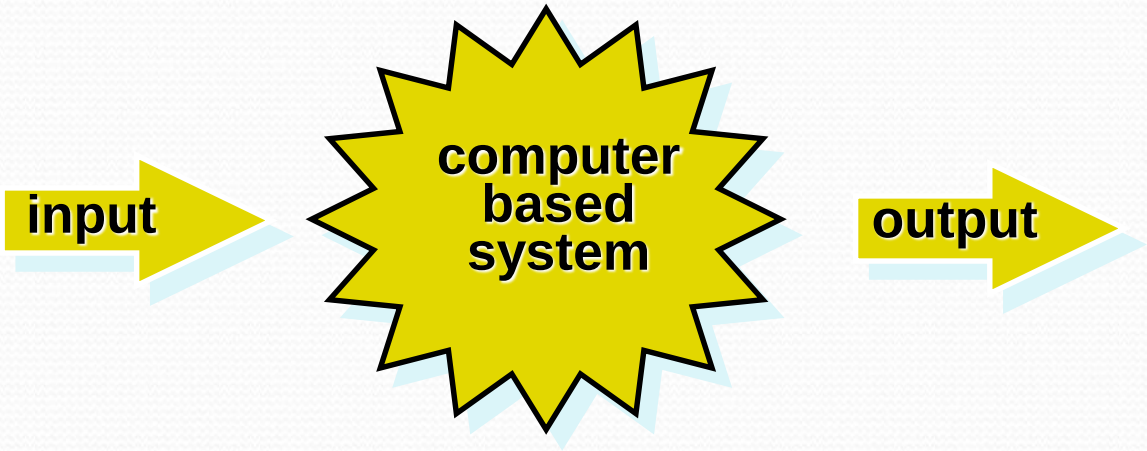
1. **Structured analysis:** An approach which considers **data and the processes that transform the data as separate entities**.
   - Data objects are modeled in a way that **defines their attributes and relationships.**
   - Processes that manipulate data objects are modeled in a manner that shows **how they transform data as data objects flow** through the system.
2. **Object Oriented analysis:** A second approach to analysis modeled, which focuses on
   - the definition of classes and
   - the manner in which they collaborate with one another to effect customer requirements.

Flow-Oriented Modeling

- Data flow modeling is a core modeling activity in structured analysis.
- Flow oriented modeling represents **how data objects are transformed** when they move through the system.
- **data flow diagram (DFD)** is the diagrammatic form that is used to represent the **data flow.**
- The purpose of DFD is to provide a semantic bridge between user and systems developers.

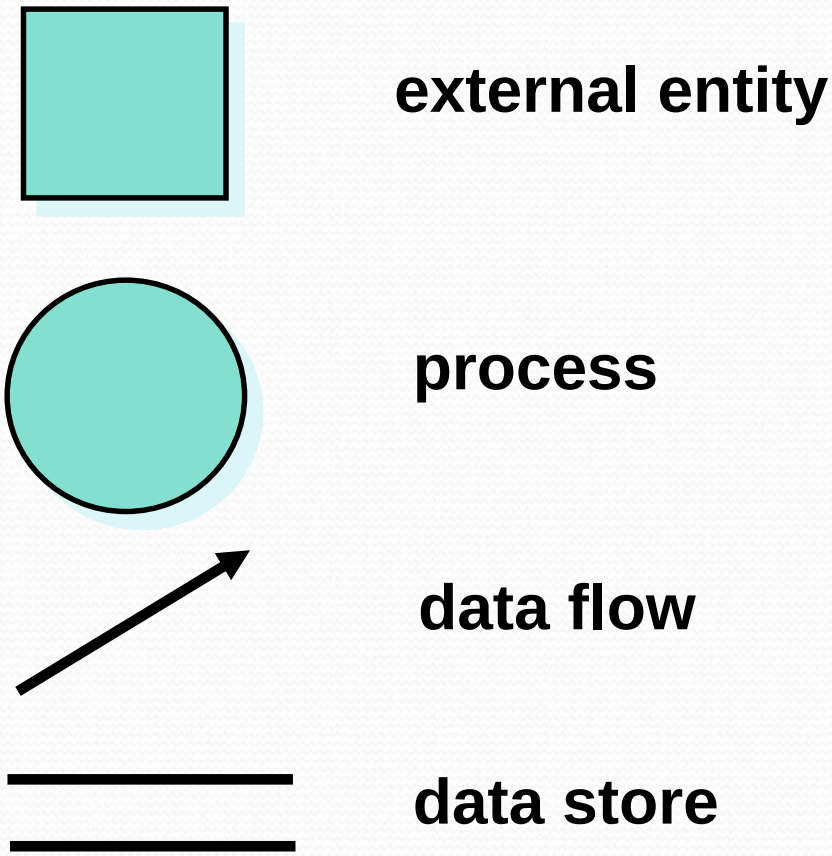
The Flow Model

Every computer-based system is an information transform ....



Creating a Data Flow Model

Flow Modeling Notations are:



External Entity

A producer or consumer of data

Examples: a person, a device, a sensor

Another example: computer-based system

Data must always originate somewhere and must always be sent to something

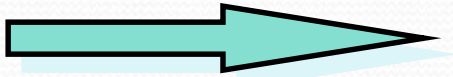
Process

A data transformer
(changes input to output)

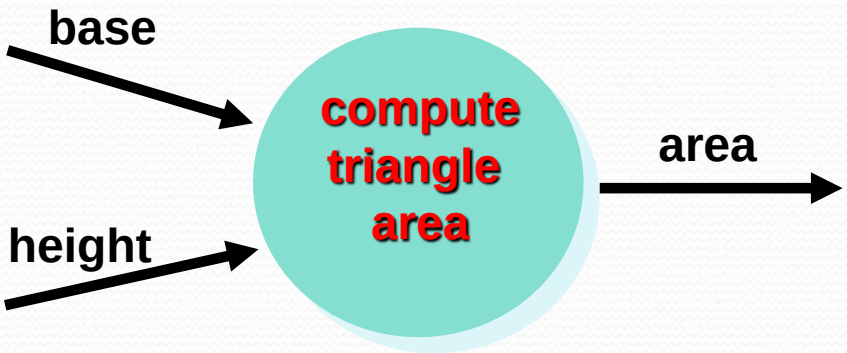
Examples: compute taxes, determine area, format report, display graph

Data must always be processed in some way to achieve system function

Data Flow

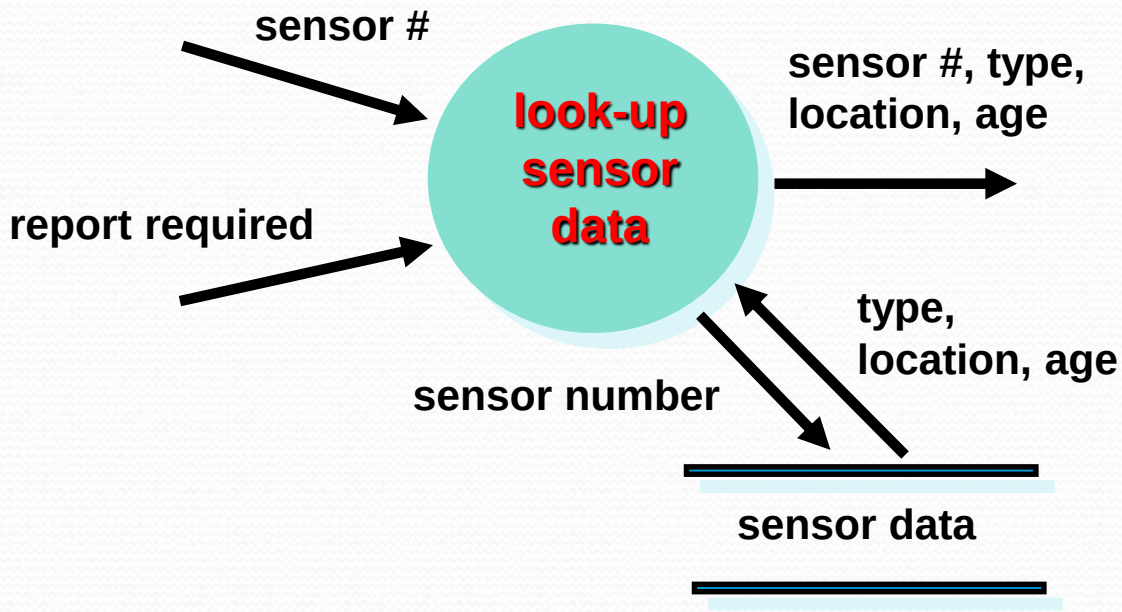


Data flows through a system, beginning as input and transformed into output.



Data Stores

Data is often stored for later use.



Creating a Data Flow Model

- DFD:

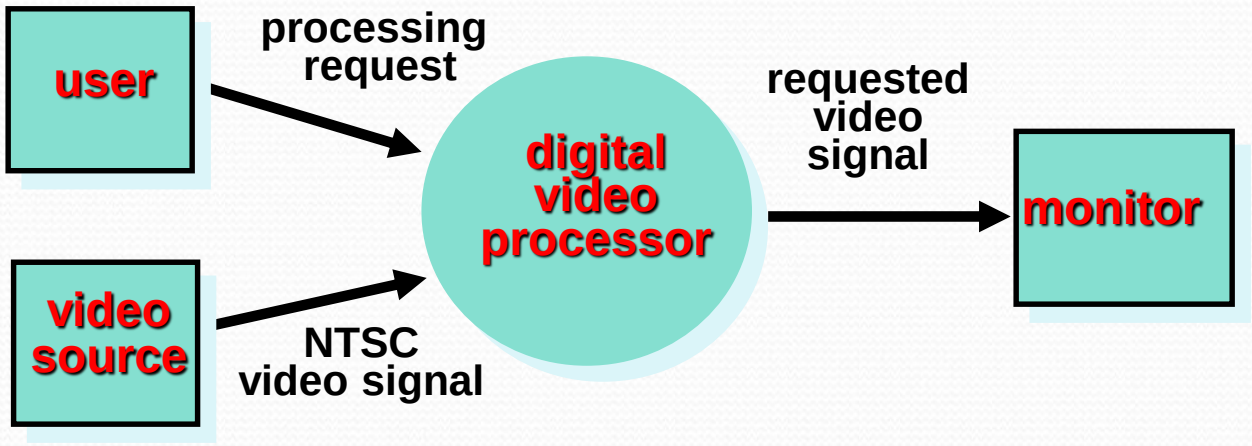
Guidelines:

1. The level 0 DFD should depict the software / system as a single bubble
2. Input and output should be clearly noted
3. Refinement should begin by isolating candidate processes, data objects, and data stores to be represented at the **next level**
4. All arrows and bubbles (processes) must be labeled with meaningful names
5. Information flow continuity must be maintained from level to level

Constructing a DFD

- Review user scenarios and/or the data model to isolate data objects and use a grammatical parse to determine “operations”
- Determine external entities (producers and consumers of data)
- Create a level 0 DFD

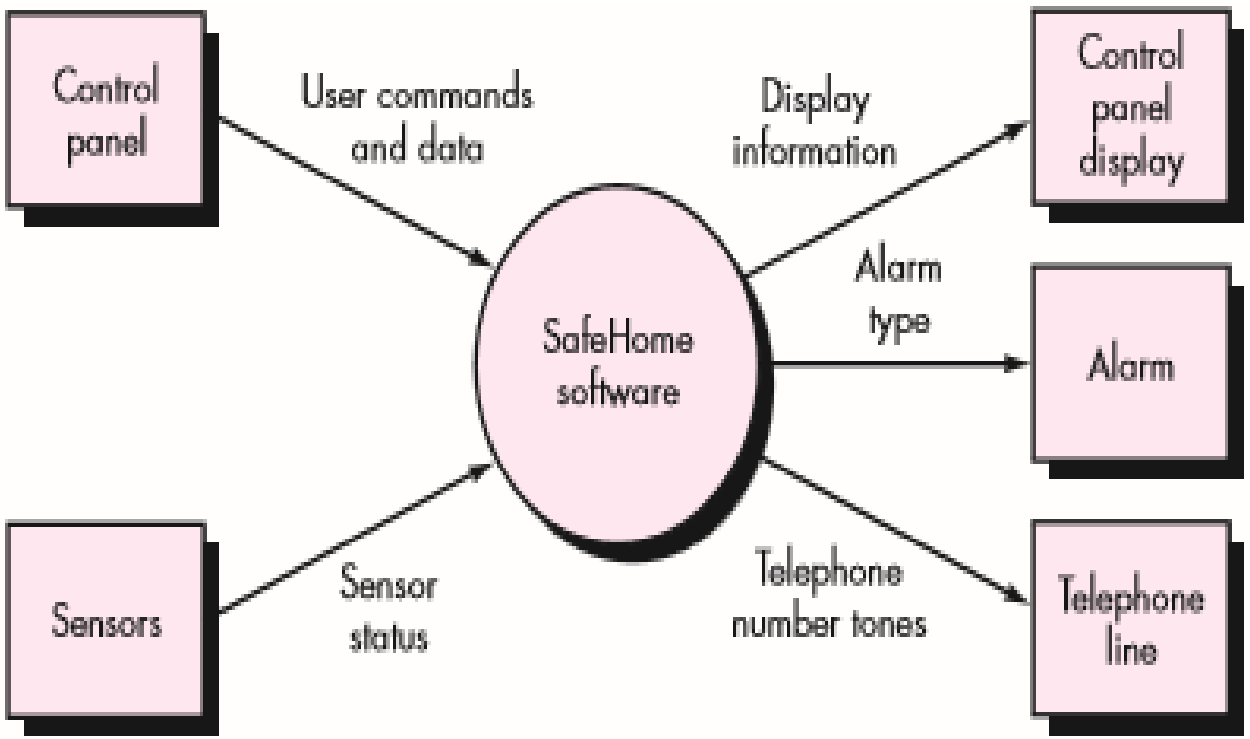
Level-0 DFD --example



Another Example for 0-DFD

FIGURE 7.1

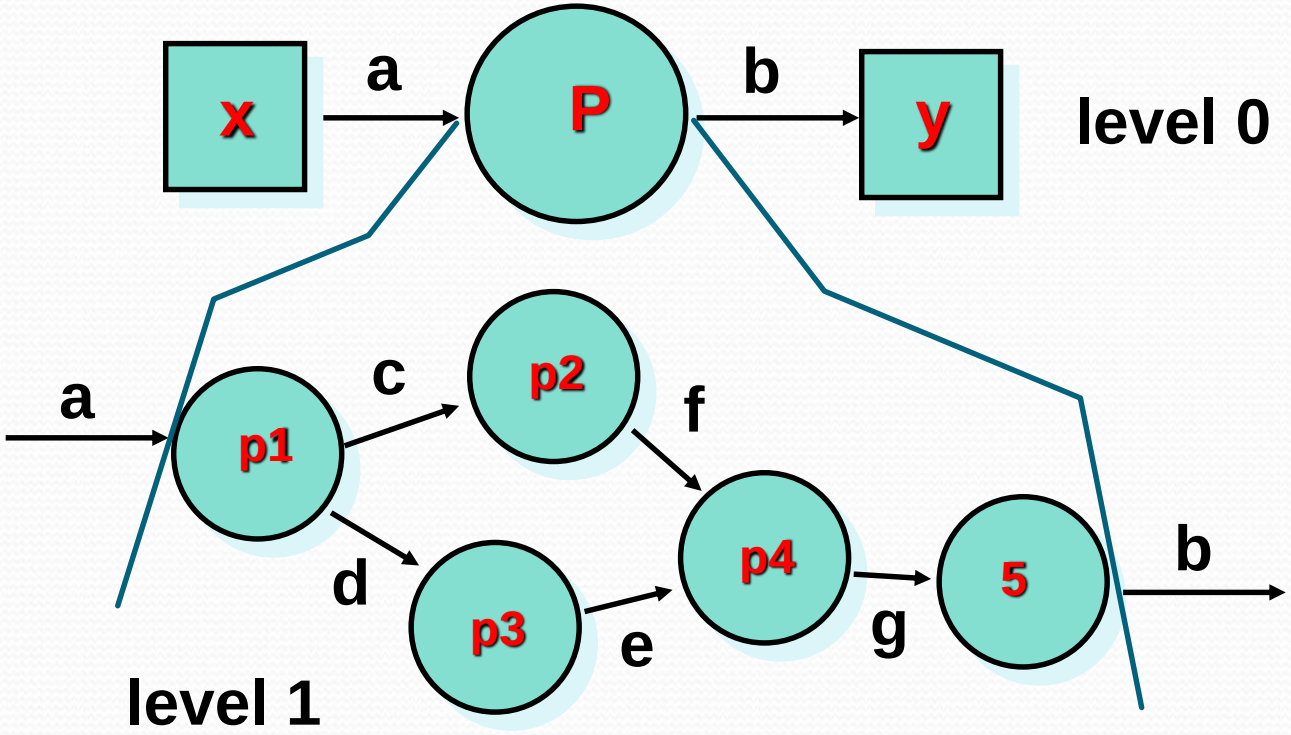
Context-level DFD for the *SafeHome* security function



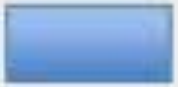
Constructing a Level -1 DFD

- Write a narrative describing the transform
- Parse to determine next level transforms
- “Balance” the flow to maintain data flow continuity
- Develop a level -1 DFD
- Use a 1:5 (approx.) expansion ratio

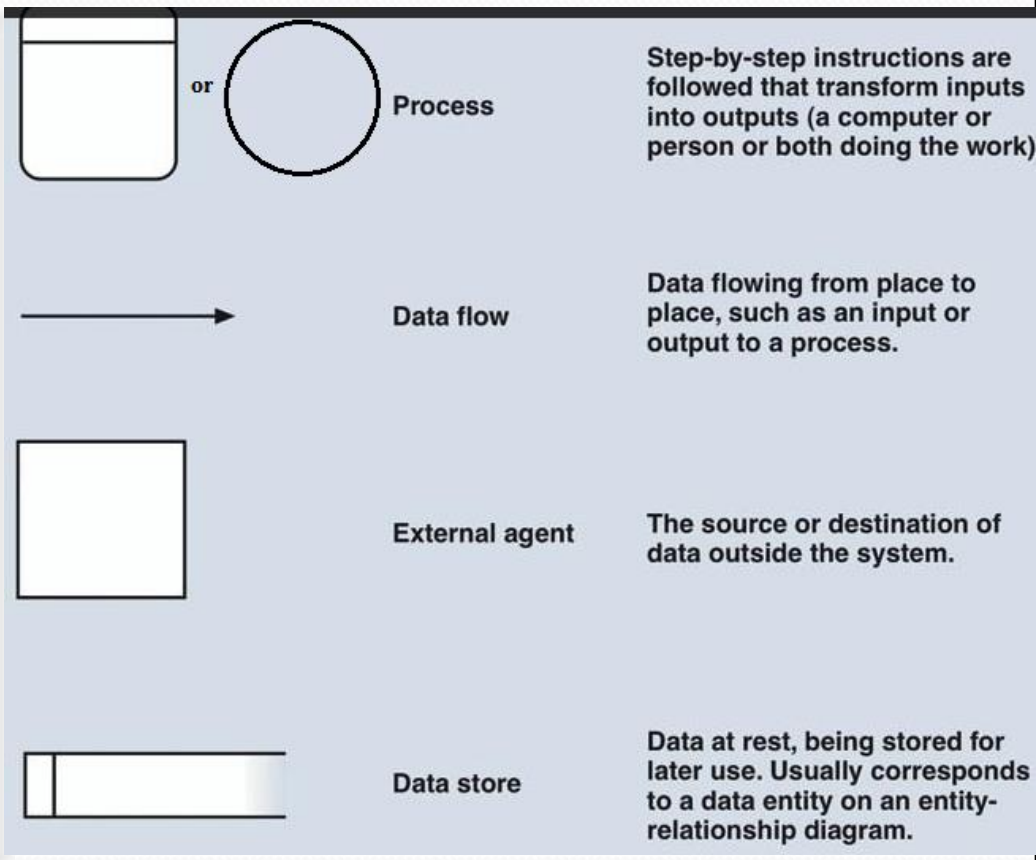
The Data Flow Hierarchy



Flow chart Vs DFD

Flow Chart Symbol	Meaning	Explanation
	Start and end	The symbol denoting the beginning and end of the flow chart.
	Step	This symbol shows that the user performs a task. (Note: In many flow charts steps and actions are interchangeable.)
	Decision	This symbol represents a point where a decision is made.
	Action	This symbol means that the user performs an action. (Note: In many flow charts steps and actions are interchangeable.)
	Flow line	A line that connects the various symbols in an ordered way.

(a) Flow Chart



(b) DFD

Summary: DFD symbols

Process

Step-by-step instructions are followed that transform inputs into outputs (a computer or person or both doing the work).

Data flow

Data flowing from place to place, such as an input or output to a process.

External agent

The source or destination of data outside the system.

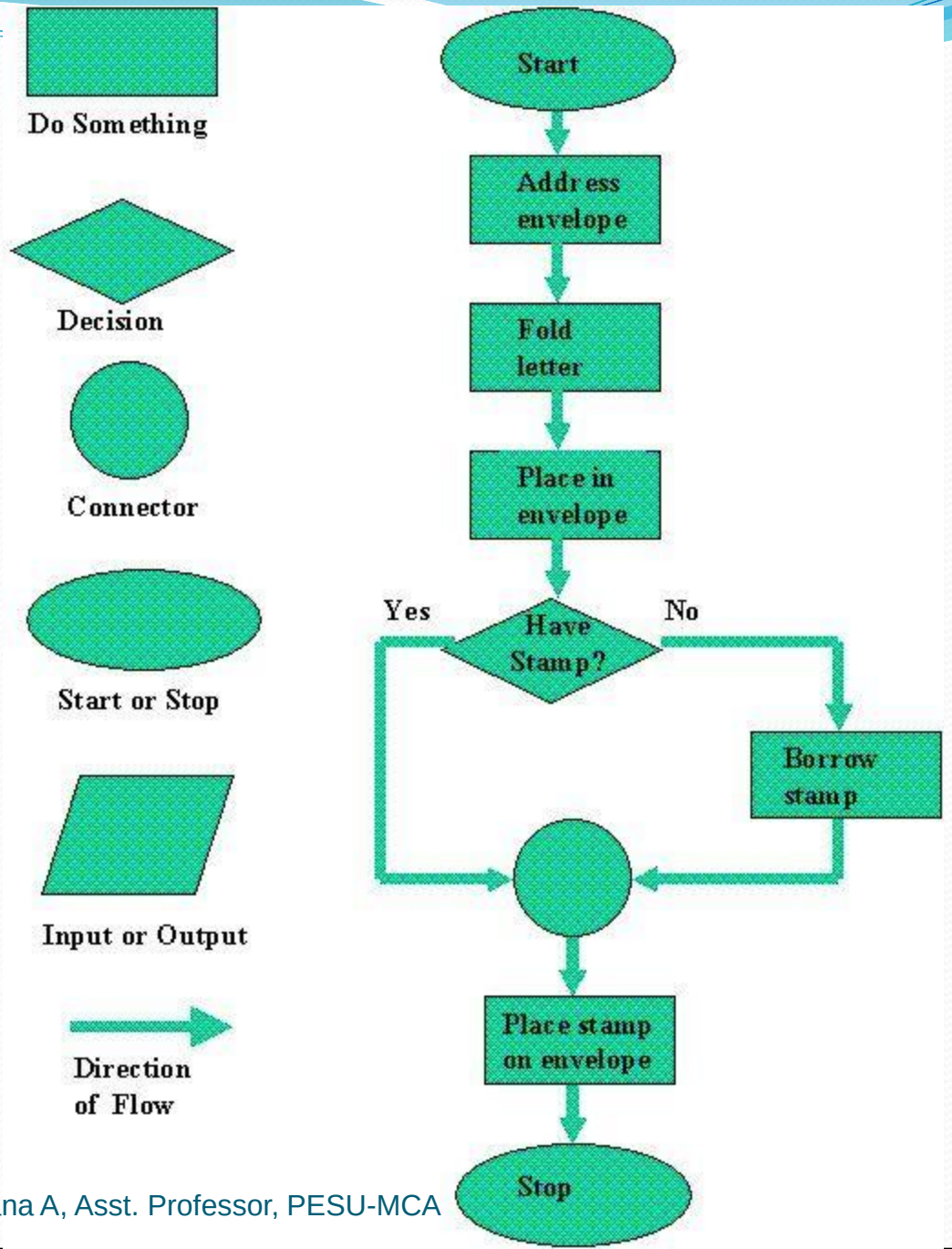
Data store

Data at rest, being stored for later use. Usually corresponds to a data entity on an entity-relationship diagram.

Real-time link

Communication back and forth between an external agent and a process as the process is executing (e.g., credit card verification).

Flow chart Symbols



Flow Modeling summary

- Each bubble is refined until it does just one thing
- The **expansion ratio decreases as the number of levels increase.**
- Most systems require between 3 and 7 levels for an adequate flow model
- A single data flow item (arrow) may be expanded as levels increase (data dictionary provides information)

Behavioral Modeling

- The behavioral model indicates **how software will respond to external events or stimuli.**
- To **create the model**, the analyst must perform the following steps:
  - **Evaluate all use-cases** to fully understand the sequence of interaction within the system.
  - **Identify events** that drive the interaction sequence and understand how these events relate to specific objects.
  - **Create a sequence** for each use-case.
  - **Build a state diagram** for the system.
  - **Review the behavioral model** to verify accuracy and consistency.

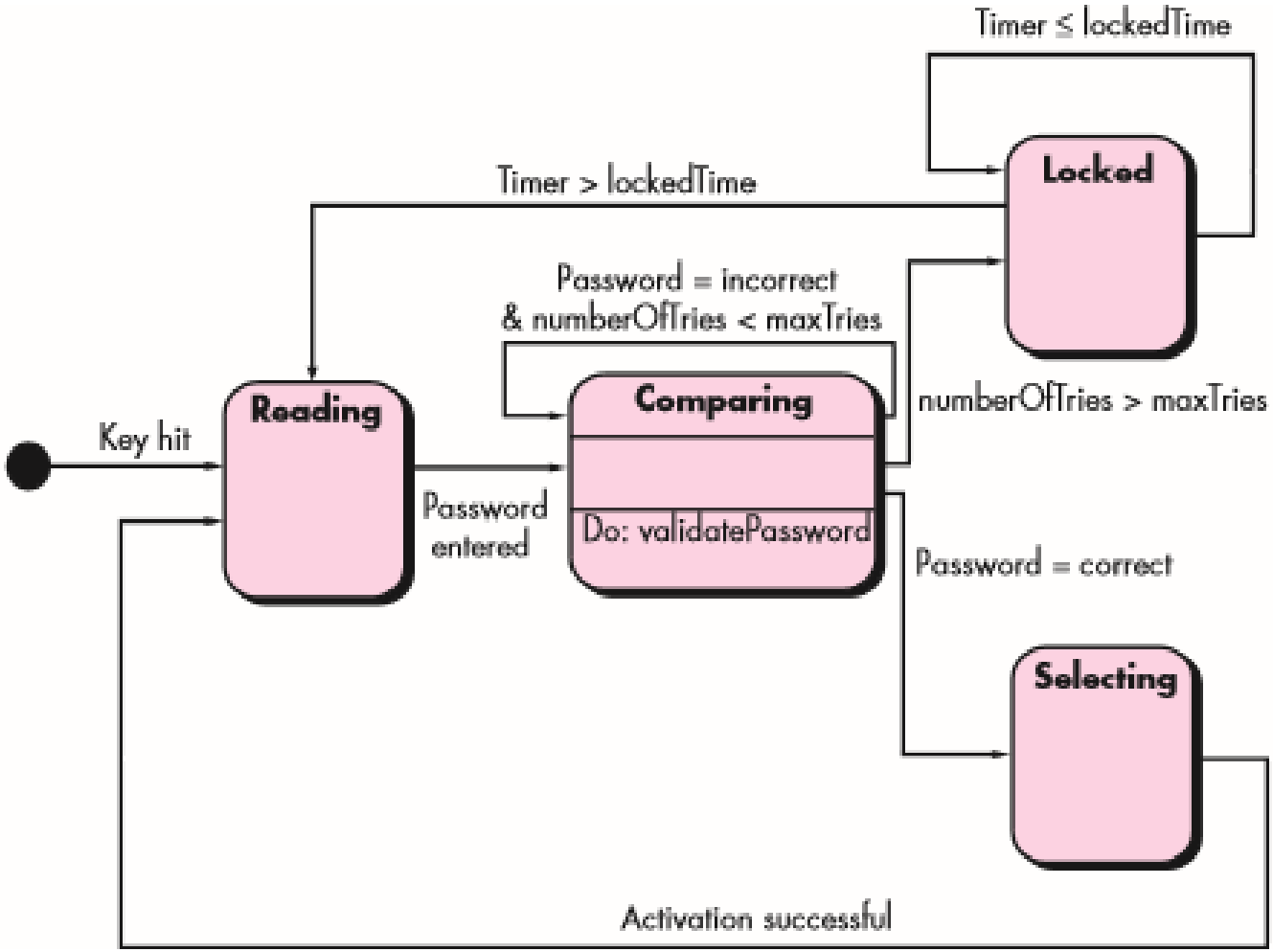
Behavioral Modeling…

- make a list of the different states of a system (How does the system behave?)
- indicate how the system makes a transition from one state to another (How does the system change state?)
  - indicate event
  - indicate action
- draw a state diagram or a sequence diagram

State Representations

- In the context of behavioral modeling, two different characterizations of states must be considered:
  - the state of each class as the system performs its function and
  - the state of the system as observed from the outside as the system performs its function
- The state of a class takes on both passive and active characteristics.
  - A *passive state* is simply the **current status of all of an object's attributes**.
  - The *active state* of an object indicates the current status of the object as it **undergoes a continuing transformation** or **processing**.

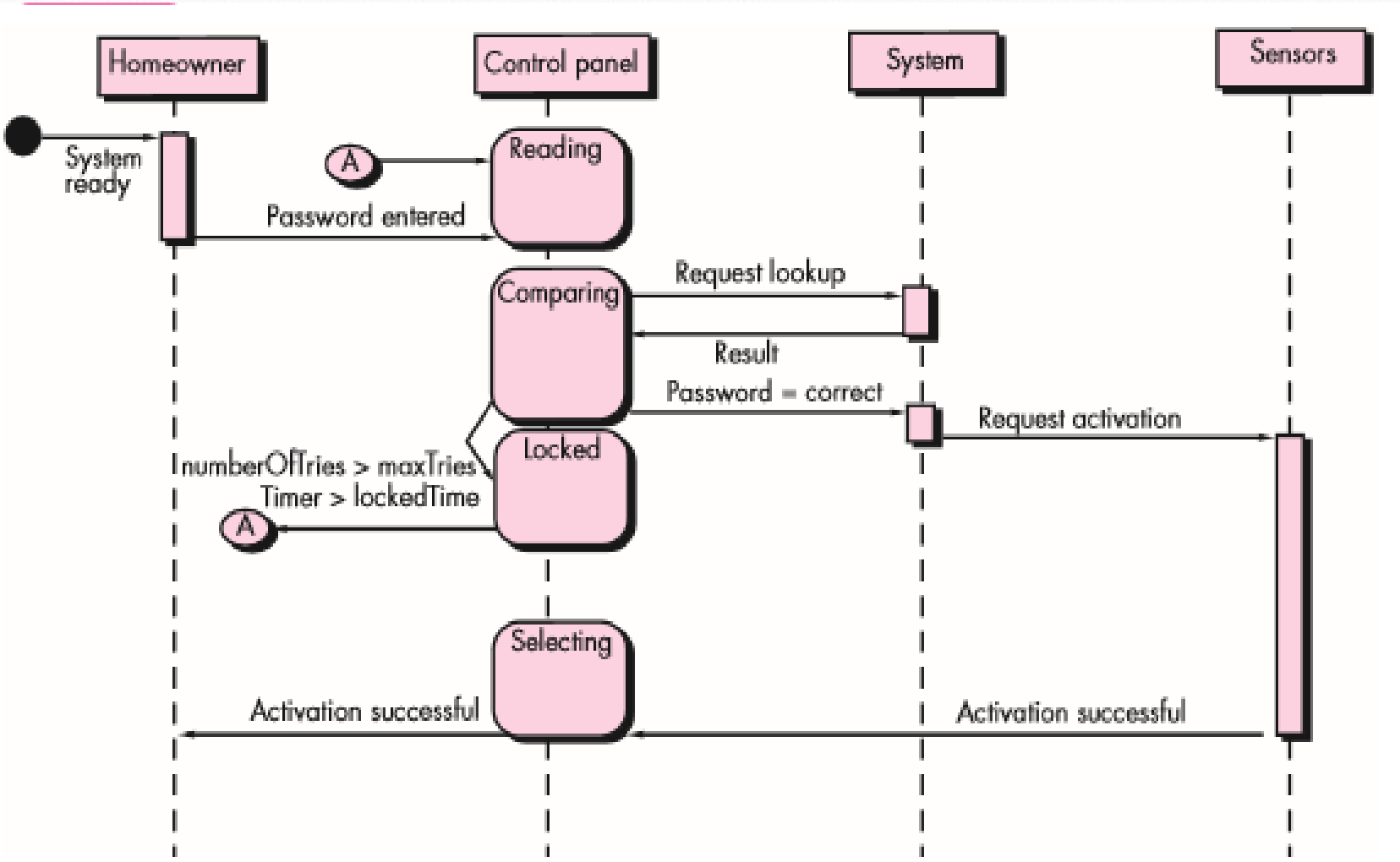
Ex: State Diagram for the ControlPanel Class



The States of a System

- **state**—a set of observable circumstances that characterizes the behavior of a system at a given time
- **state transition**—the movement from one state to another
- **event**—an occurrence that causes the system to exhibit some predictable form of behavior
- **action**—process that occurs as a consequence of making a transition

Sequence Diagram for SafeHome Security function



Patterns for Requirements Modeling

- Software patterns are a mechanism for capturing domain knowledge in a way that allows it to be **reapplied when a new problem is encountered**
  - domain knowledge can be applied to a new problem within the same application domain
  - the domain knowledge captured by a pattern can be applied by analogy to a completely different application domain.
- The original author of an analysis pattern does not "create" the pattern, but rather, *discovers* it as requirements engineering work is being conducted.
- Once the pattern has been discovered, it is documented

Discovering Analysis Patterns

- The most **basic element** in the description of a requirements model is the **use case**.
- A coherent set of use cases may serve as the basis for discovering one or more analysis patterns.
- A ***semantic analysis pattern*** (SAP) "is a pattern that describes a small set of coherent use cases that together describe a basic generic application."

END OF UNIT 2

Ms. Archana A, Asst.
Professor, PESU-MCA