SOFTWARE ENGINEERING

Week 4

Requirements Engineering and Analysis

Assoc. Prof. Ayşe TOSUN Prof. Tolga OVATMAN

Istanbul Technical University
Computer Engineering Department

Agenda

1. Requirements Engineering
   1. Requirement Engineering Processes
   2. Case Study
2. Requirements Analysis
   1. **Data Model:** Database objects and relations
   2. **Functional Model:** Data flow
   3. **Behavioural Model:** Control flow, Events and states

1. Requirements Engineering
2. Requirements Analysis

Requirements Engineering

4.1

Requirement Engineering

- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.
  - The current situation
  - The functionality that the new system should support
  - The environment in which the system will be deployed
  - Deliverables expected by the client
  - Delivery dates
  - A set of acceptance criteria

Types of Requirement

- User requirements
  - Statements in natural language plus diagrams of the services that the system provides and its operational constraints. Written for customers.
- System requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

User and System Requirements

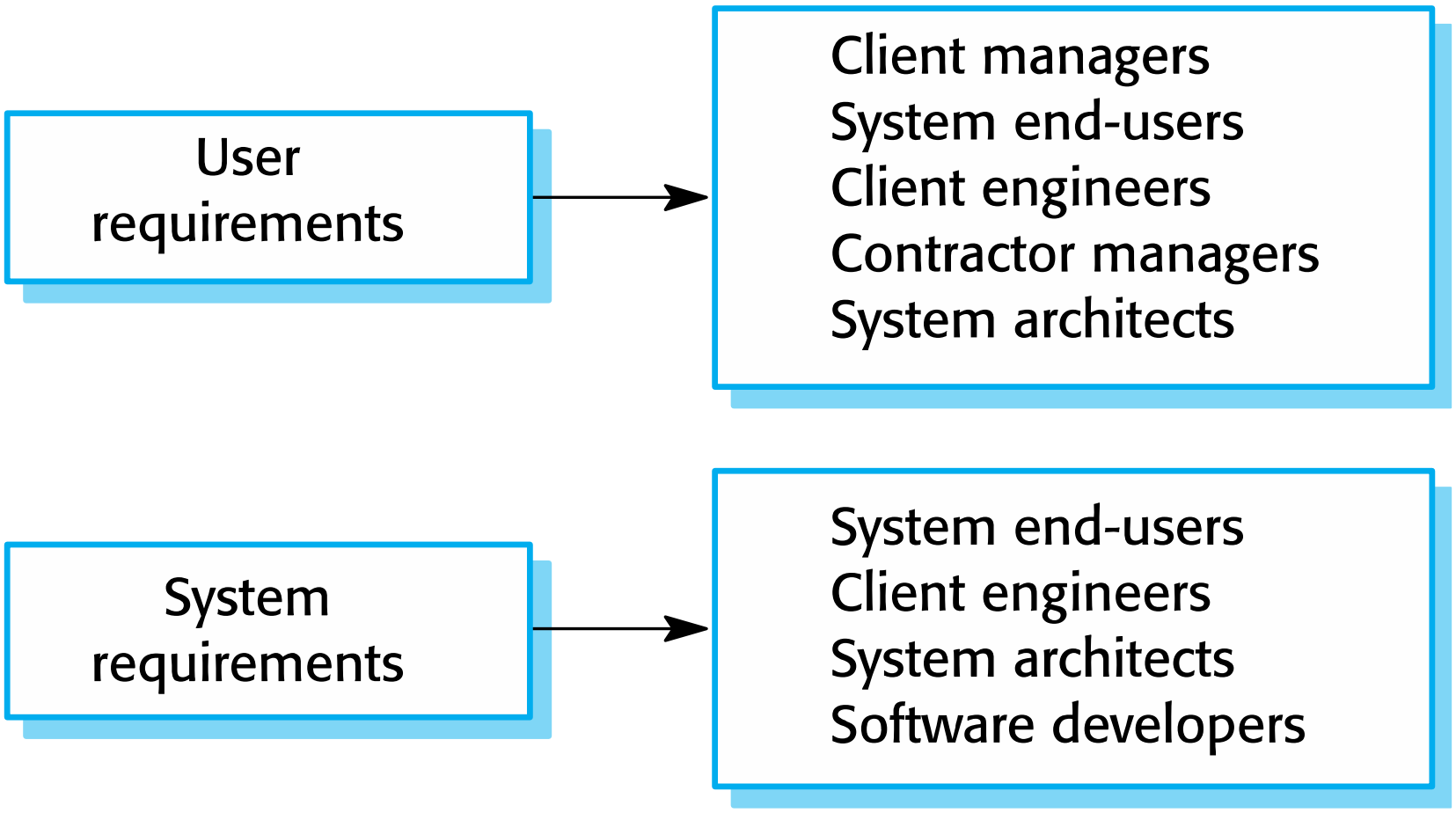
User requirement definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20 mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Users of Requirements



Functional and Non-functional Requirements

- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.
- Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.
- Domain requirements
  - Constraints on the system from the domain of operation

Example Requirements

- Calculate the discount at the rate of 14% for a customer spending amount
- Number of significant digits to which accuracy should be maintained in all numerical calculations is 4
- A book can be deleted from the Library Management System by the Database Administrator only

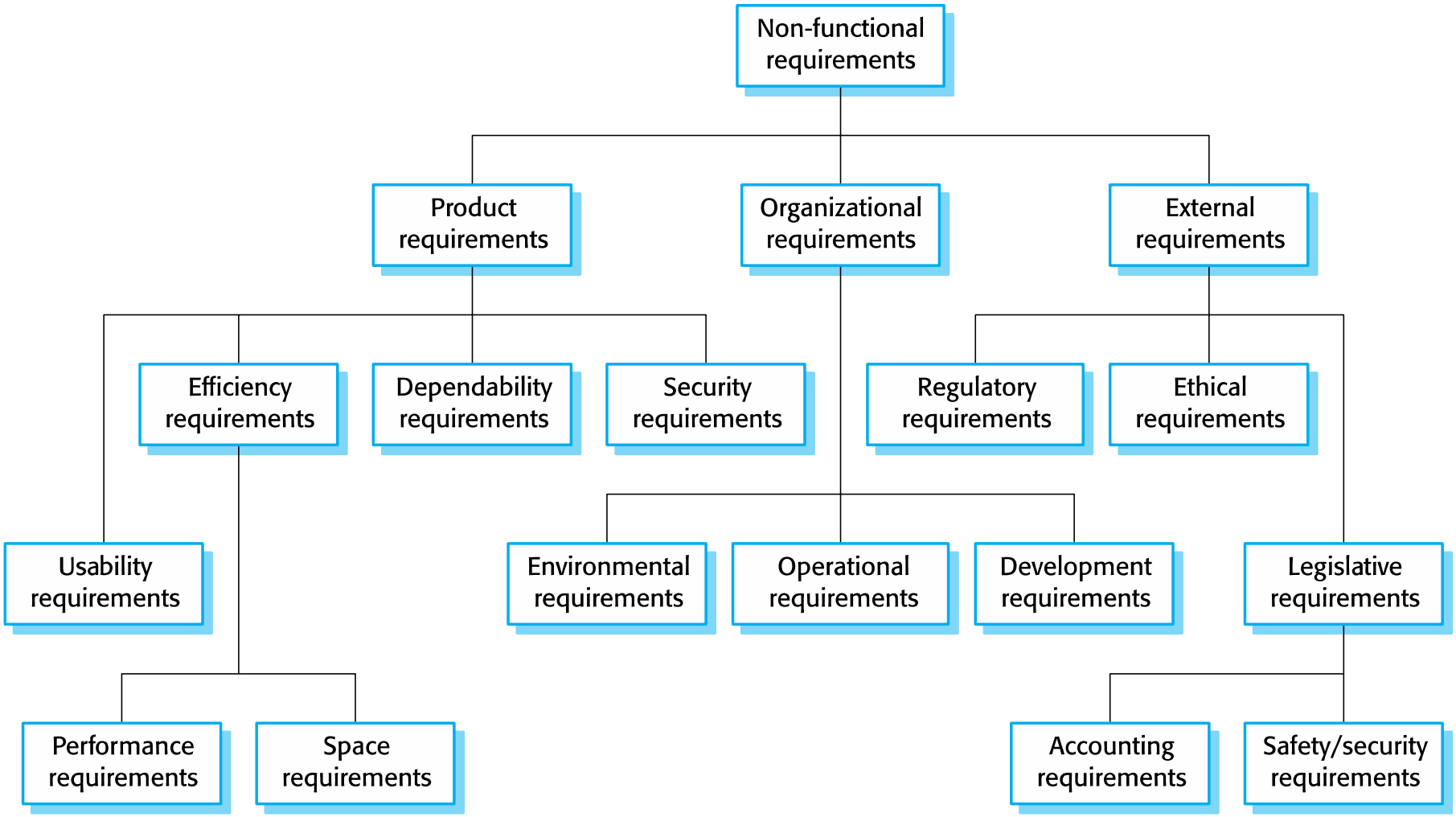
---

- The response time of the system should always be less than 5seconds
- The software should operate on a UNIX based system
- Experienced officers should be able to use all the system functions after a total training of two hours. After this training, the average number of errors made by experienced officers should not exceed two per day

What is usually NOT in requirements?

- System structure, implementation technology
- Development methodology
- Development environment
- Implementation language
- Reusability

Types of Non-functional Requirements



Requirements Specification

- The process of writing down the user and system requirements in a requirements document.
- User requirements have to be understandable by end-users and customers who do not have a technical background.
- System requirements are more detailed requirements and may include more technical information.
- The requirements may be part of a contract for the system development
  - It is therefore important that these are as complete as possible.

Guidelines For Writing Requirements

- Invent a standard format and use it for all requirements.
- Assign a unique number and source (mostly people) for each requirement.
- Use language in a consistent way. Use *shall* for mandatory requirements, *should* for desirable requirements.
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of computer jargon.
- Include an explanation (rationale) of why a requirement is necessary.

Example SRS Documents

- https://slcc.pressbooks.pub/technicalwritingatslcc/chapter/software-requirements-specification-srs/
- https://personal.utdallas.edu/~chung/SP/RequirementsAnalysisDocumentTemplate.htm
- https://www.utdallas.edu/~chung/RE/Presentations07S/Team_1_Doc/Documents/SRS4.0.doc

Proper Documentation of Requirements

- Requirements are analyzed and validated against the following critical attributes:
  - **Complete**: Reflect the system objectives and specify the relationship between the software and the rest of the subsystems
  - **Unambiguous:** One and only one interpretation
  - **Traceable**: Unique identifier allowing the software design, code, test to be precisely traced back to requirements
  - **Consistent**: No conflicts.
  - **Feasible**: Analyzed if there is a question on its feasibility
  - **Uniquely Identified**: For traceability and testability
  - **Design Free**: Describe what software needs to accomplish, NOT how.
  - **Using 'Shall' and Related Words**: *Shall* – must be implemented. *Should*, *May* – nonbinding provisions. *Will* – declaration of purpose.

Al Florence, *Reducing Risks Through Proper Specification of Software Requirements*, MITRE Corp, April 2002, https://personal.utdallas.edu/~chung/RE/florence.pdf

Requirements Modeling

Distinguish the *environment*, the *wish* list from the *customer's perspective* and the *behaviour* of the software system

- **Domain properties**
  - Before the switch is moved to the On position, the user must add ground coffee to the filter and insert it in the coffee machine.
  - Before the switch is moved to the On position, the user must add water to the reservoir.
- **Requirements**
  - When the user moves the three-way switch to the On position, coffee shall be brewed.
- **Specification**
  - If the three-way switch is On, the coffee brewer shall be actuated

C. Gunter, E. Gunter, M. Jackson and P. Zave, "A Reference Model for Requirements and Specifications," IEEE Software, May/June 2000. pp. 37-43

Requirements Modeling v2

- **Domain properties**
  - There will always be a nurse close enough to hear the buzzer
  - The sound from the heart falling below a certain threshold indicates that heart has (is about to) stop
- **Requirements**
  - A warning system notifies the nurse if the patient's heartbeat stops
- **Specification**
  - If the sound from the sensor falls below a certain threshold, the buzzer shall be actuated

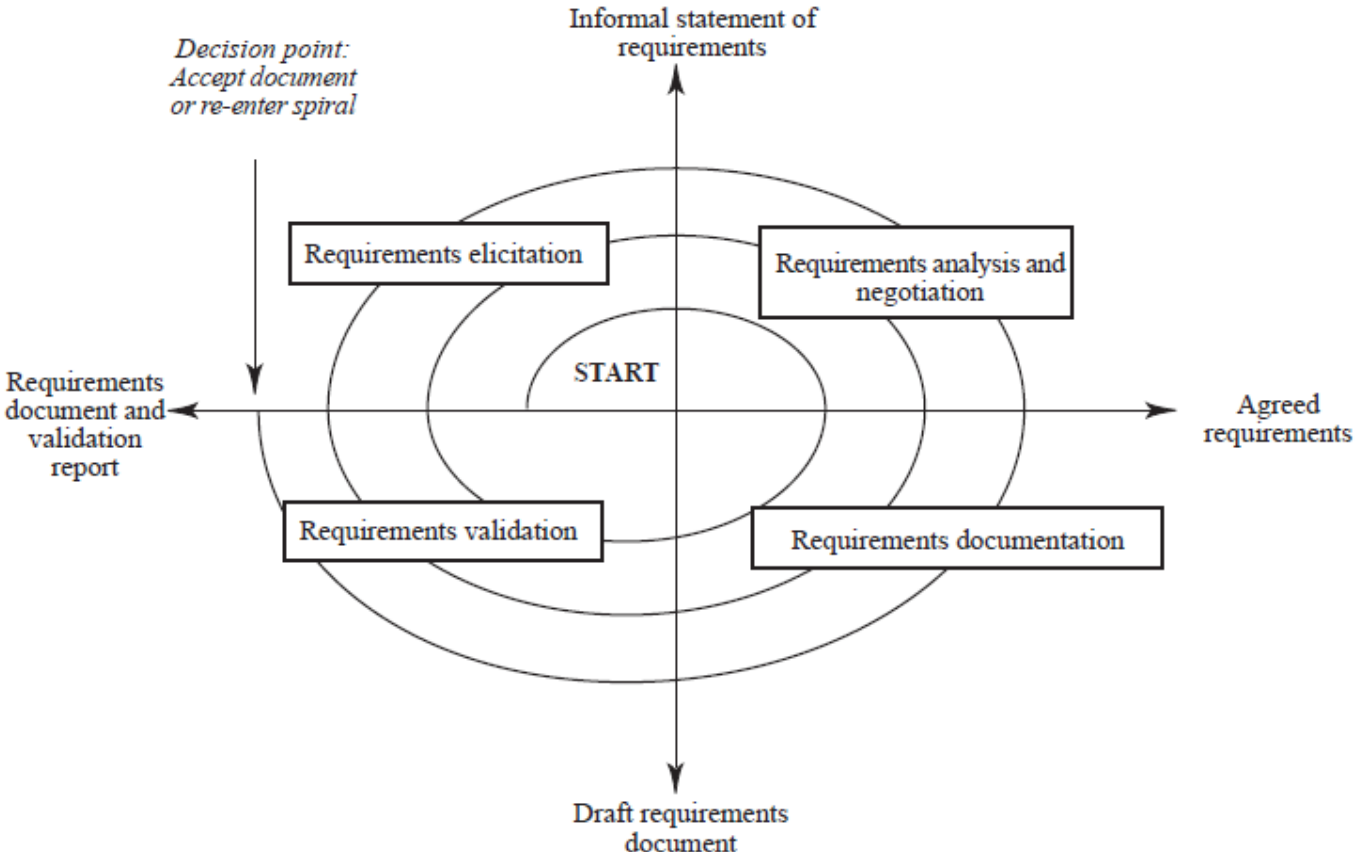
C. Gunter, E. Gunter, M. Jackson and P. Zave, "A Reference Model for Requirements and Specifications," IEEE Software, May/June 2000. pp. 37-43

1. Requirements Engineering
2. Requirements Analysis

Requirements Engineering Processes

4.1.1

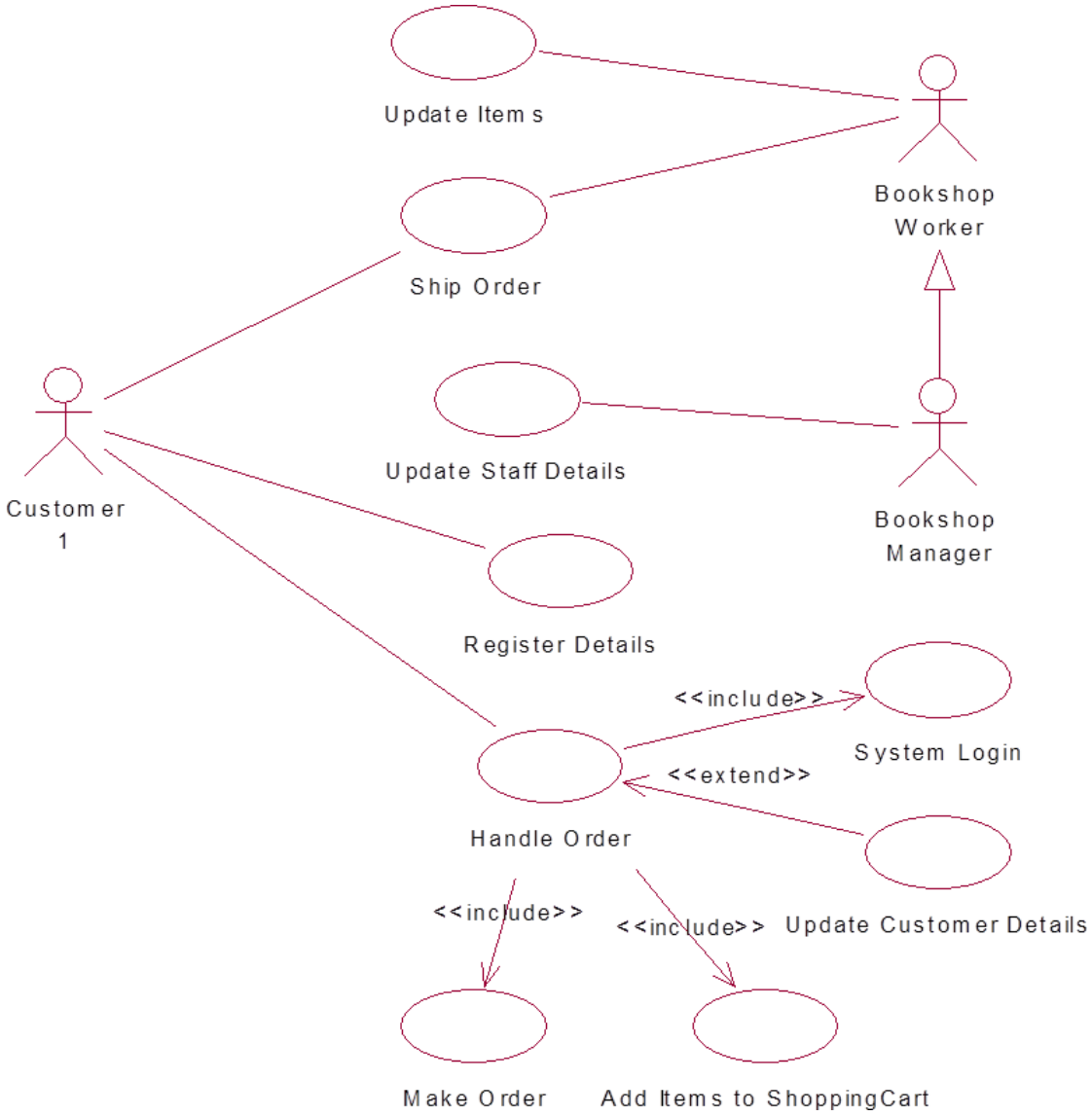
A Spiral View Of The Requirements Engineering Process



Use Cases

- Use case diagrams are used to visualize, specify, construct, and document the (intended) behavior of the system, during requirements capture and analysis.
- Provide a way for developers, domain experts and end-users to Communicate.
- Serve as basis for testing.
- Main authors: *Booch, Rumbaugh, and Jacobson*
- The Object Management Group (OMG) is responsible for standardization. (www.omg.org)
- Current version is UML 2.5

Example : Use Case Diagram



Example : Use Case (Money Withdraw) - I

- **Use Case:** Withdraw Money
- **Author:** ZB
- **Date:** 1-OCT-2004
- **Purpose:** To withdraw some cash from user's bank account
- **Overview:** The use case starts when the customer inserts his credit card into the system. The system requests the user PIN. The system validates the PIN. If the validation succeeded, the customer can choose the withdraw operation else alternative 1 – validation failure is executed. The customer enters the amount of cash to withdraw. The system checks the amount of cash in the user account, its credit limit. If the withdraw amount in the range between the current amount + credit limit the system dispense the cash and prints a withdraw receipt, else alternative 2 – amount exceeded is executed.
- **Cross References:** R1.1, R1.2, R7

Example : Use Case (Money Withdraw) - II

- **Actors:** Customer
- **Pre Condition:**
  - The ATM must be in a state ready to accept transactions
  - The ATM must have at least some cash on hand that it can dispense
  - The ATM must have enough paper to print a receipt for at least one transaction
- **Post Condition:**
  - The current amount of cash in the user account is the amount before the withdraw minus the withdraw amount
  - A receipt was printed on the withdraw amount
  - The withdraw transaction was audit in the System log file

Example : Use Case (Money Withdraw) - III

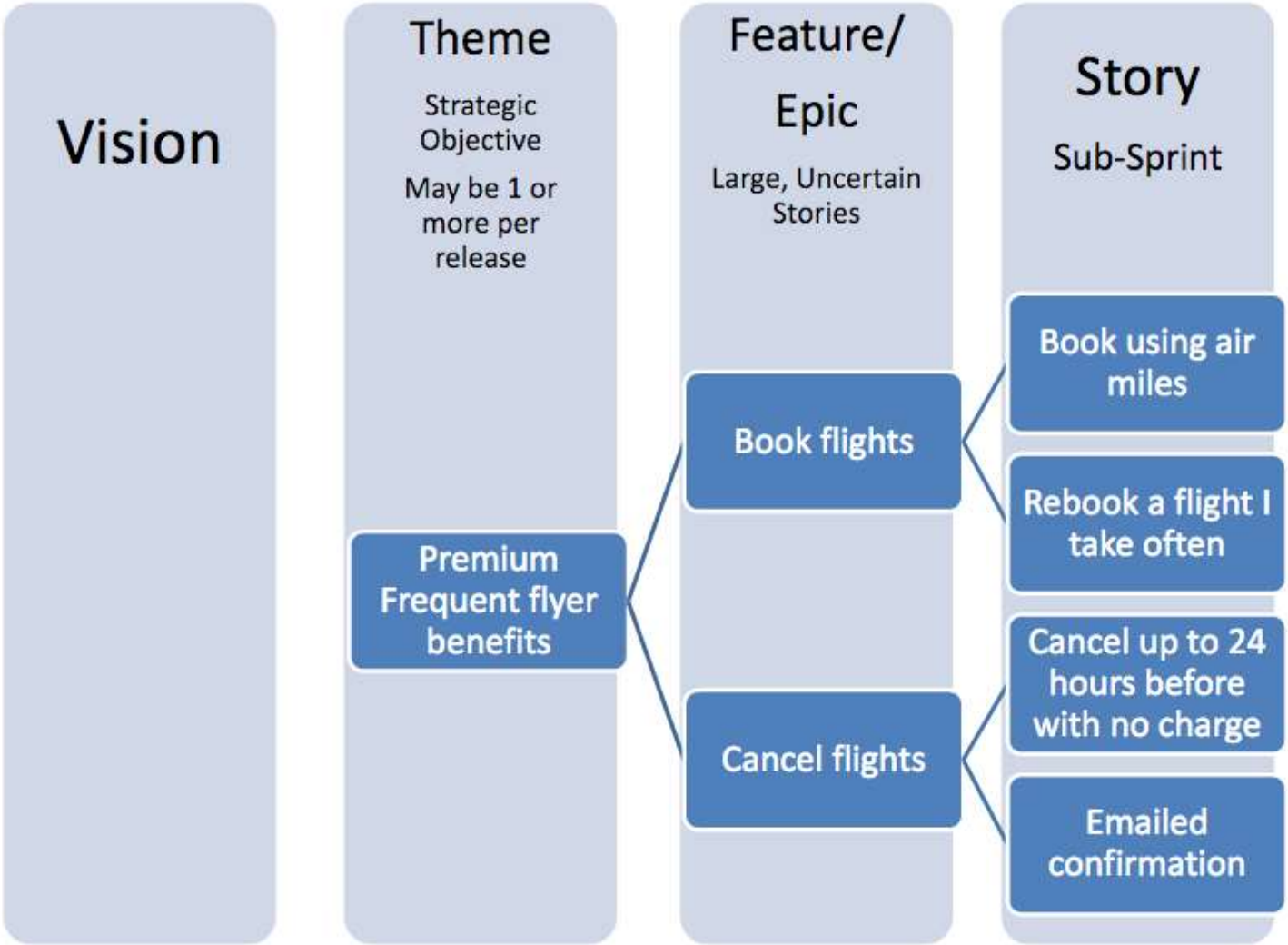
Typical Course of events:

Actor Actions	System Actions
1. Begins when a Customer arrives at ATM	
2. Customer inserts a Credit card into ATM	3. System verifies the customer ID and status
5. Customer chooses "Withdraw" operation	4. System asks for an operation type
7. Customer enters the cash amount	6. System asks for the withdraw amount
	8. System checks if withdraw amount is legal
	9. System dispenses the cash
	10. System deduces the withdraw amount from account
	11. System prints a receipt
13. Customer takes the cash and the receipt	12. System ejects the cash card

Example : Use Case (Money Withdraw) - IV

- **Alternative flow of events:**
  - Step 3: Customer authorization failed. Display an error message, cancel the transaction and eject the card.
  - Step 8: Customer has insufficient funds in its account. Display an error message, and go to step 6.
  - Step 8: Customer exceeds its legal amount. Display an error message, and go to step 6.
- **Exceptional flow of events:**
  - Power failure in the process of the transaction before step 9, cancel the transaction and eject the card

Feature/Epic/User Story



What is a User Story?

- Simple, Clear, short description of customer valued functionality
- User Stories are an eXtreme Programming technique
- This may optionally be used to capture Product Backlog Items
- The Product Backlog is a Scrum Artifact
- User Stories capture Who, What and Why of any requirement
- 3Cs – Card, Conversation, Confirmation
- Conversation rather than documentation

User Story Template

Title: Priority:

As a [type of user], I want [goal] so that

[Value]

Notes:
Assumptions:
Constraints:

Estimate:

User Story Example

Checkout Using Credit Card

Priority: 25

As a book shopper, I can checkout using my credit card
So that I can purchase a selected book.

Notes: Support mc, visa, amex

Constraints: Must use SBI payment gateway

Estimate: 13pts

Gherkin Format

- Given [context]
- When [some event]
- Then [outcome]

- As a user I want emails with attachments to go faster so that I can work more efficiently

- As a user I want to cancel a reservation so that I avoid being charged full rate

Gherkin Format

- **Scenario:** Scenario explanation {Operation Name}
- **Given** {some, a} user{s} {with/out} something
- **When** the user asks for present status
- **Then** a {positive, negative} answer should return
- **And** response time should be lower than {duration} milliseconds

```
public class MyStepdefs {
        @Given("I have (\\d+) cukes in my belly")
        public void I_have_cukes_in_my_belly(int cukes) {
                System.out.format("Cukes: %n\n", cukes);
        } }
```

TDD Example

FizzBuzz consists of:

1. Printing every number from 1 to 100, with the following exceptions.
2. If the number is evenly divisible by both three and five, then the word “FizzBuzz” should be printed instead of the number.
3. If the number is evenly divisible by three, then the word “Fizz” should be printed instead of the number.
4. If the number is evenly divisible by five, then the word “Buzz” should be printed instead of the number.

TDD Example

```
public class FizzBuzz {

    private static String fizzbuzzify(int num) {
        return "";
    }

    public static void main(String args[]) {

    }

}
```

```
public class FizzBuzzTest {

    @Test
    public void test1Returns1() {
        String returnedVal = FizzBuzz.fizzbuzzify(1);
        assertEquals("1", returnedVal);
    }

}
```

```
public class FizzBuzz {

    private static String fizzbuzzify(int num) {
        return "1";
    }

    public static void main(String args[]) {

    }

}
```

TDD Example

```
public class FizzBuzzTest {

    @Test
    public void test1Returns1() {
        String returnedVal = FizzBuzz.fizzbuzzify(1);
        assertEquals("1", returnedVal);
    }

    public void test2Returns2() {
        String returnedVal = FizzBuzz.fizzbuzzify(2);
        assertEquals("2", returnedVal);
    }

}
```

```
public class FizzBuzz {

    private static String fizzbuzzify(int num) {
        if (num == 1) {
            return "1";
        } else {
            return "2";
        }
    }

    public static void main(String args[]) {

    }

}
```

```
public class FizzBuzz {

    private static String fizzbuzzify(int num) {

        return String.valueOf(num);
    }

    public static void main(String args[]) {

    }

}
```

TDD Example

```java
@Test
public void test3ReturnsFizz() {
    String returnedVal = FizzBuzz.fizzbuzzify(3);
    assertEquals("Fizz", returnedVal);
}
```

```java
private static String fizzbuzzify(int num) {
    if (num % 3 == 0) {
        return "Fizz";
    } else {
        return String.valueOf(num);
    }
}
```

```java
@Test
public void test5ReturnsBuzz() {
    String returnedVal = FizzBuzz.fizzbuzzify(5);
    assertEquals("Buzz", returnedVal);
}
```

```java
private static String fizzbuzzify(int num) {
    if (num % 3 == 0) {
        return "Fizz";
    } else if (num % 5 == 0) {
        return "Buzz";

    } else {
        return String.valueOf(num);
    }
}
```

TDD Example

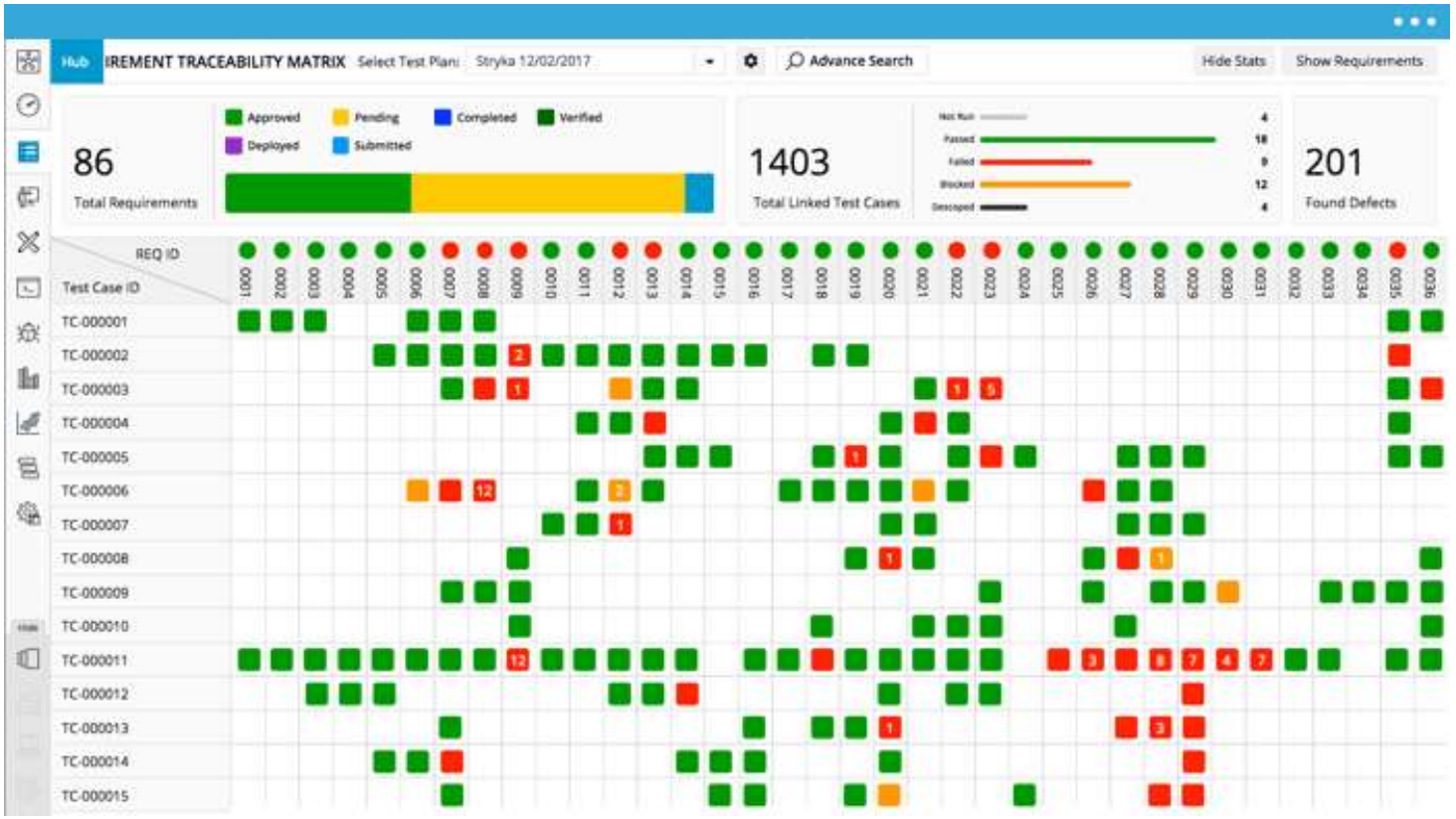
```java
@Test
public void test15ReturnsFizzBuzz() {
    String returnedVal = FizzBuzz.fizzbuzzify(15);
    assertEquals("FizzBuzz", returnedVal);
}
```

```java
private static String fizzbuzzify(int num) {
    if ((num % 3 == 0) && (num % 5 == 0)) {
        return "FizzBuzz";
    } else if (num % 3 == 0) {
        return "Fizz";
    } else if (num % 5 == 0) {
        return "Buzz";
    } else {
        return String.valueOf(num);
    }
}
```

Traceability

Req. ID	Requirement Description	Justification	Test Case ID	Test Result	Notes
1	Landing page	Starting point and first impression with site visitors	Test01, Test02	Pass	
2	Login	Lets account holders access their information, prompts new visitors to create an account	Test04, Test05, Test06	Fail	Users can't move beyond the login page's CAPTCHA
3	Email unsubscribe button	CAN-SPAM Act requirement	Test07	Not executed	

Traceability



Scenarios

- "A narrative description of what people do and experience as they try to make use of computer systems and applications" [M. Carrol, Scenario-based Design, Wiley, 1995]

- A concrete, focused, informal description of a single feature of the system used by a single actor.

- Scenarios can have many different uses during the software lifecycle
  - *Requirements Elicitation*: As-is scenario, visionary scenario
  - *Client Acceptance Test:* Evaluation scenario
  - *System Deployment:* Training scenario.

Types of Scenarios

- As-is scenario:
  - Used in describing a current situation. Usually used in re-engineering projects. The user describes the system.
    - Example: Description of Letter-Chess
- Visionary scenario:
  - Used to describe a future system. Usually used in greenfield engineering and reengineering projects.
  - Can often not be done by the user or developer alone
    - Example: Description of an interactive internet-based Tic Tac Toe game tournament.
- Evaluation scenario:
  - User tasks against which the system is to be evaluated.
    - Example: Four users (two novice, two experts) play in a Tic Tac Toe tournament in ARENA.
- Training scenario:
  - Step by step instructions that guide a novice user through a system
    - Example: How to play Tic Tac Toe in the ARENA Game Framework.

How do we find scenarios?

- Don’t expect the client to be verbal if the system does not exist (greenfield engineering)
- Don’t wait for information even if the system exists
- Engage in a dialectic approach (evolutionary, incremental engineering)
  - You help the client to formulate the requirements
  - The client helps you to understand the requirements
  - The requirements evolve while the scenarios are being developed

Heuristics for finding Scenarios

- Ask yourself or the client the following questions:
  - What are the primary tasks that the system needs to perform?
  - What data will the actor create, store, change, remove or add in the system?
  - What external changes does the system need to know about?
  - What changes or events will the actor of the system need to be informed about?
- However, don't rely on *questionnaires* alone.
- Insist on *task observation* if the system already exists (interface engineering or reengineering)
  - Ask to speak to the end user, not just to the software contractor
  - Expect resistance and try to overcome it

1. Requirements Engineering
2. Requirements Analysis

Case Study

4.1.2

Example: Accident Management System

- What needs to be done to report a “Cat in the Tree” incident?
- What do you need to do if a person reports “Warehouse on Fire?”
- Who is involved in reporting an incident?
- What does the system do, if no police cars are available? If the police car has an accident on the way to the “cat in a tree” incident?
- What do you need to do if the “Cat in the Tree” turns into a “Grandma has fallen from the Ladder”?
- Can the system cope with a simultaneous incident report “Warehouse on Fire?”

Example from
https://personal.utdallas.edu/~chung/SYSM6309/Scenario-Analysis.pdf

Scenario Example: Warehouse on Fire

- Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, **reports** the emergency from her car.

- Alice **enters** the **address of the building, a brief description of its location (i.e., north west corner), and an emergency level**. In addition to a fire unit, she **requests several paramedic units** on the scene given that area appear to be relatively busy. She **confirms** her input and **waits for an acknowledgment**.

- John, **the Dispatcher**, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and **acknowledges** the report. He **allocates a fire unit** and two paramedic units to the Incident site and **sends their estimated arrival time** (ETA) to Alice.

- Alice **received** the acknowledgment and the ETA.

Observations about Warehouse on Fire Scenario

- Concrete scenario
  - Describes a single instance of reporting a fire incident.
  - Does not describe all possible situations in which a fire can be reported.

- Participating actors
  - Bob, Alice and John

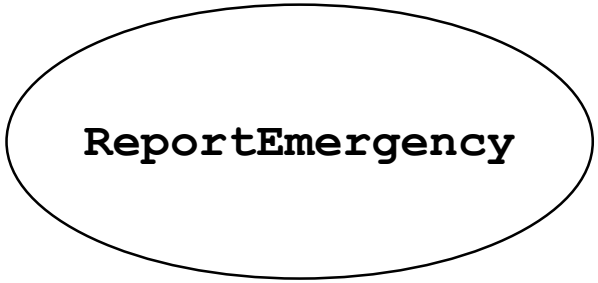
Next goal, after the scenarios are formulated:

- Find all the use cases in the scenario that specifies all possible instances of how to report a fire
  - Example: “Report Emergency “ in the first paragraph of the scenario is a candidate for a use case

- Describe each of these use cases in more detail
  - Participating actors
  - Describe the Entry Condition
  - Describe the Flow of Events
  - Describe the Exit Condition
  - Describe Exceptions
  - Describe Special Requirements (Constraints, Nonfunctional Requirements

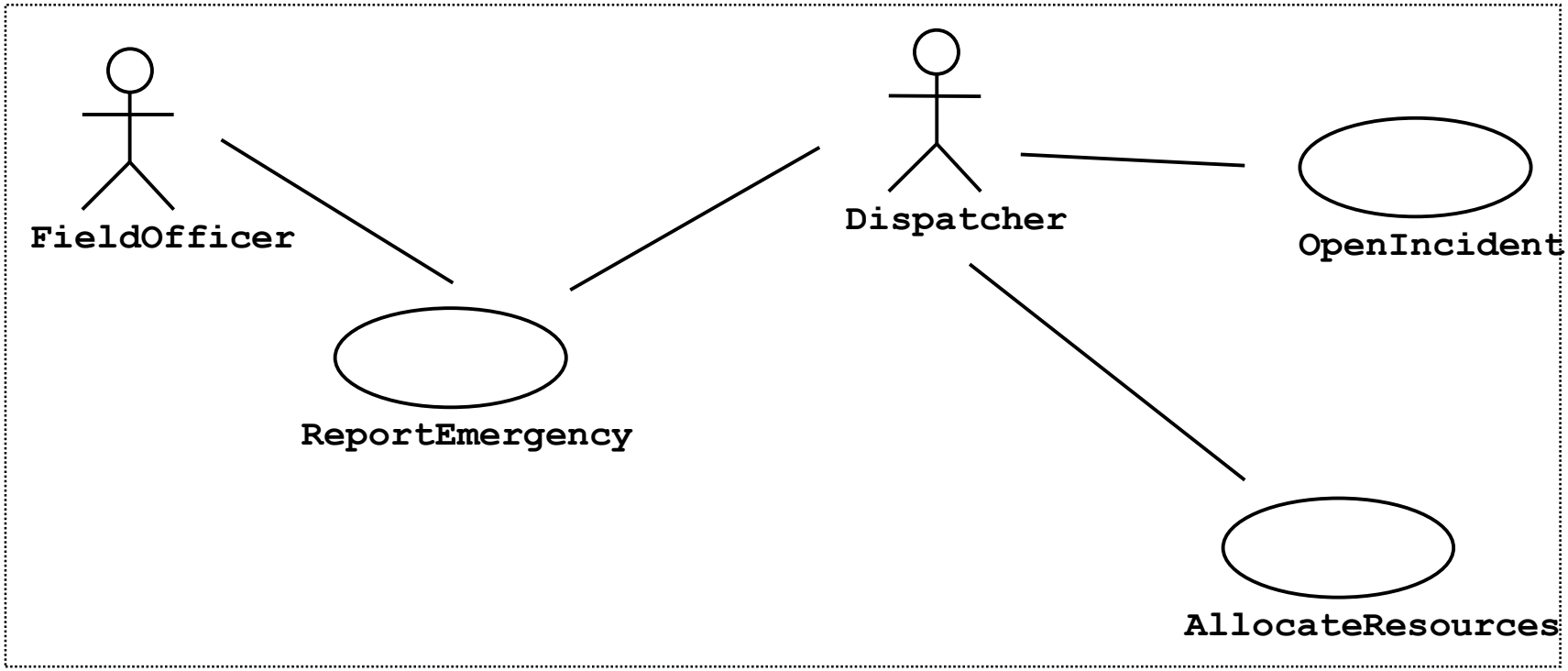
Use Cases

- A use case is a flow of events in the system, including interaction with actors
- It is initiated by an actor
- Each use case has a name
- Each use case has a termination condition
- Graphical Notation: An oval with the name of the use case



Use Case Model: **The set of all use cases specifying the complete functionality of the system**

Example: Use Case Model for Incident Management



Heuristics: How do I find use cases?

- Select a narrow vertical slice of the system (i.e. one scenario)
  - Discuss it in detail with the user to understand the user's preferred style of interaction
- Select a horizontal slice (i.e. many scenarios) to define the scope of the system.
  - Discuss the scope with the user
- Use illustrative prototypes (mock-ups) as visual support
- Find out what the user does
  - Task observation (Good)
  - Questionnaires (Bad)

Use Case Example: ReportEmergency

- Use case name: ReportEmergency
- Participating Actors:
  - Field Officer (Bob and Alice in the Scenario)
  - Dispatcher (John in the Scenario)
- Exceptions:
  - The FieldOfficer is notified immediately if the connection between her terminal and the central is lost.
  - The Dispatcher is notified immediately if the connection between any log in FieldOfficer and the central is lost.
- Flow of Events:
  - **(next slide)**
- Special Requirements:
  - The FieldOfficer's report is acknowledged within **30 seconds**. The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.

Use Case Example: ReportEmergency Flow of Events

- The **FieldOfficer** activates the "Report Emergency" function of her terminal. FRIEND responds by presenting a form to the officer.
- The FieldOfficer fills the form, by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form, at which point, the **Dispatcher** is notified.
- The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the emergency report.
- The FieldOfficer receives the acknowledgment and the selected response.

Another Use Case Example: Allocate a Resource

- Actors:
  - *Field Supervisor:* This is the official at the emergency site….
  - *Resource Allocator:* The Resource Allocator is responsible for the commitment and de-commitment of the Resources managed by the FRIEND system.
  - *Dispatcher:* A Dispatcher enters, updates, and removes Emergency Incidents, Actions, and Requests in the system. The Dispatcher also closes Emergency Incidents.
  - *Field Officer:* Reports accidents from the Field

Another Use Case Example: Allocate a Resource

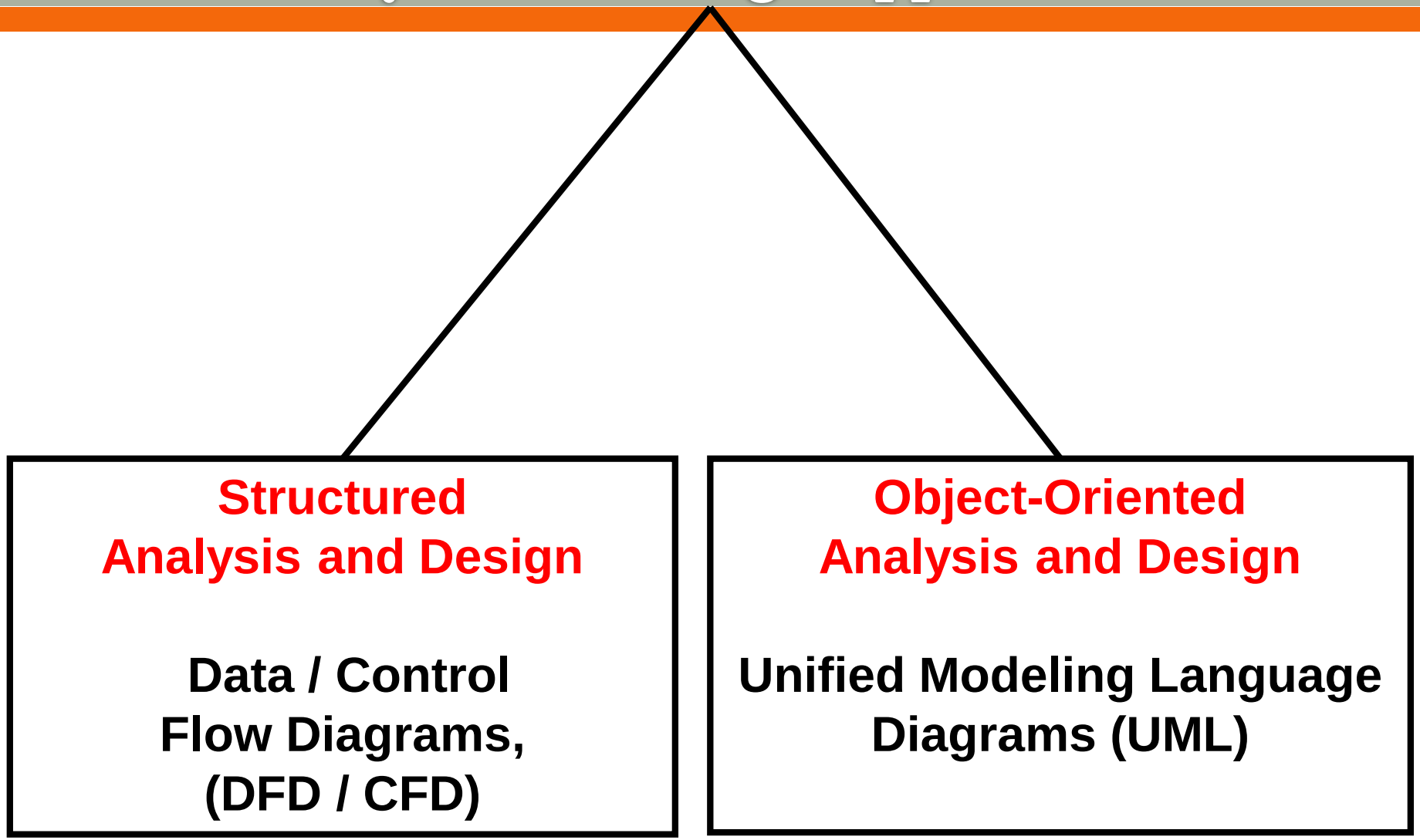
- **Use case name:** AllocateResources
- **Participating Actors:**
  - Field Officer (Bob and Alice in the Scenario)
  - Dispatcher (John in the Scenario)
  - Resource Allocator
  - Field Supervisor
- **Entry Condition**
  - The Resource Allocator has selected an available resource.
  - The resource is currently not allocated
- **Flow of Events**
  - The Resource Allocator selects an Emergency Incident.
  - The Resource is committed to the Emergency Incident.
- **Exit Condition**
  - The use case terminates when the resource is committed.
  - The selected Resource is now unavailable to any other Emergency Incidents or Resource Requests.
- **Special Requirements**
  - The Field Supervisor is responsible for managing the Resources

1. Requirements Engineering
2. Requirements Analysis 

Requirements Analysis

4.2

Analysis and Design Approaches



Elements of Analysis Model

- The Statement of Software Scope provides the basis for analysis modelling.
- The following models are built during analysis:

1. Data model: Database objects and relations
2. Functional model: Data flow
3. Behavioural model: Control flow, Events and states

Modeling the Data Domain

- Define data objects
- Establish data relationships
- Specify data content

Modelling the Functions

- Basic Idea:
  - Software transforms data
  - To achieve this, it must perform at least three generic functions: input, processing, output
  - Identify functions that transform data objects
- Begin with a context level diagram (level 0)
- Continue with more functional details in refined levels until all system functionality is represented

Modeling the Behaviour

- Basic Idea:
  - Most software responds to events from the outside world
  - This characteristic forms the basis of the behavioral model
  - A computer program always exists in some state: an externally observable mode of behaviour (e.g. waiting, computing, printing, polling) that is changed only when some event occurs
- Indicate different states of the system
- Specify events that cause the system to change state

1. Requirements Engineering
2. Requirements Analysis

Structural Analysis

4.2

1. Requirements Engineering
2. Requirements Analysis 

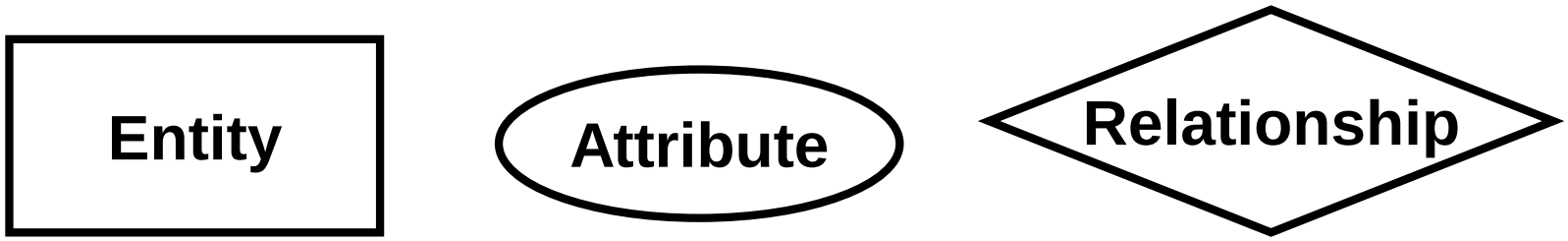
The Data Model

4.2.1

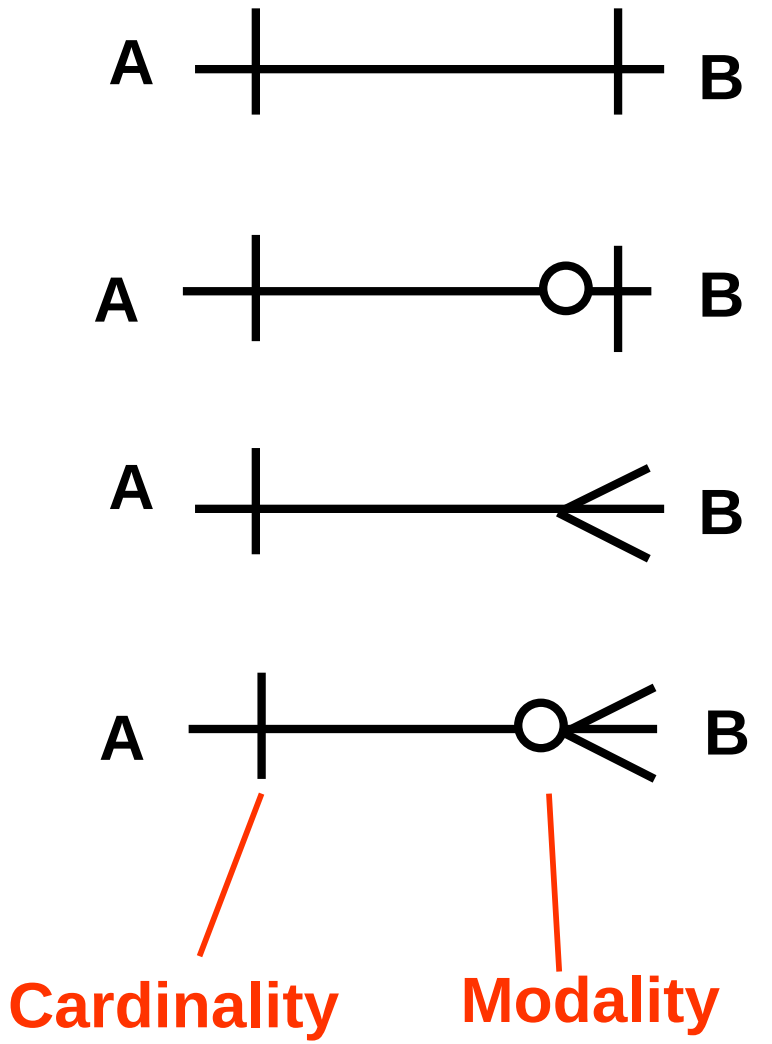
The Data Model

- Data modelling is also called Database Modelling.
- In data modelling, Entity-Relationship Diagrams are used.
- Also a data dictionary is defined for important data items.

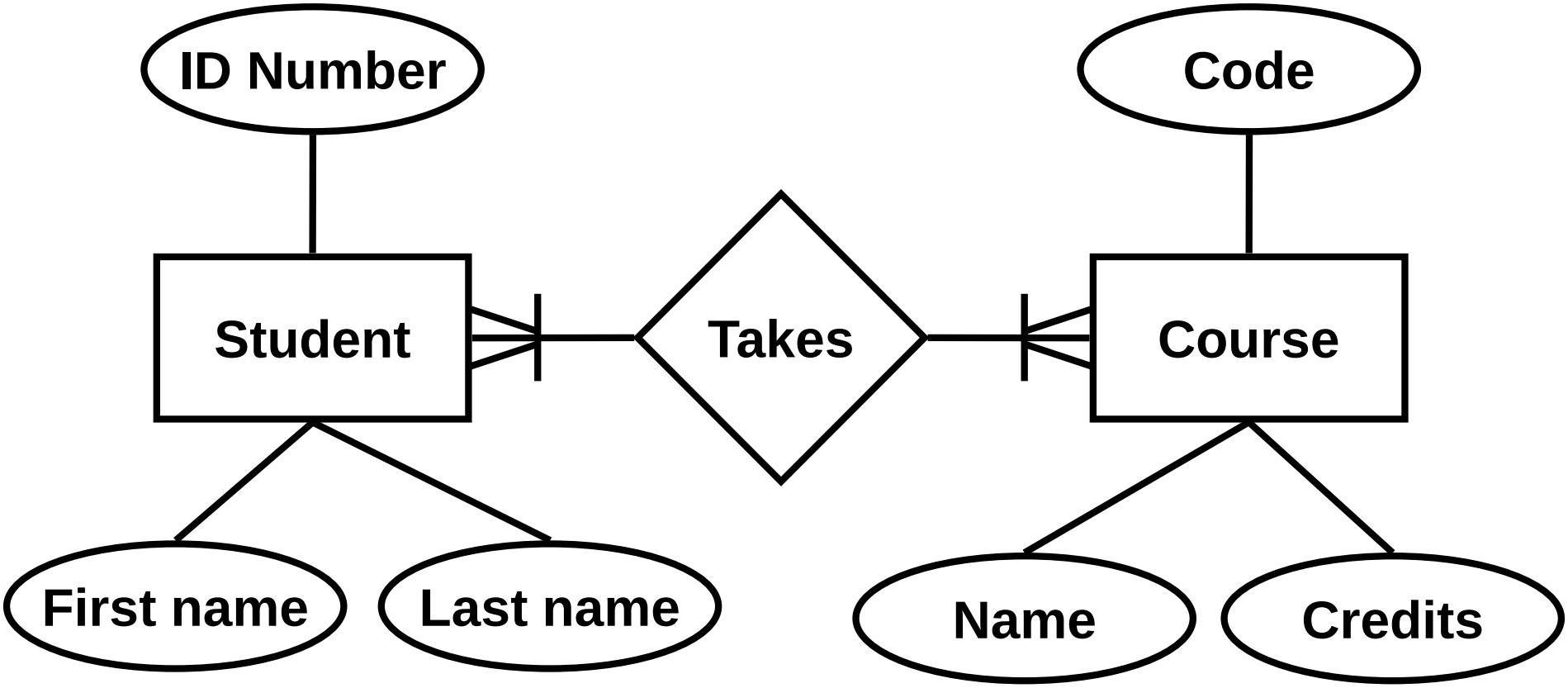
Entity Symbols (Bachman notation)



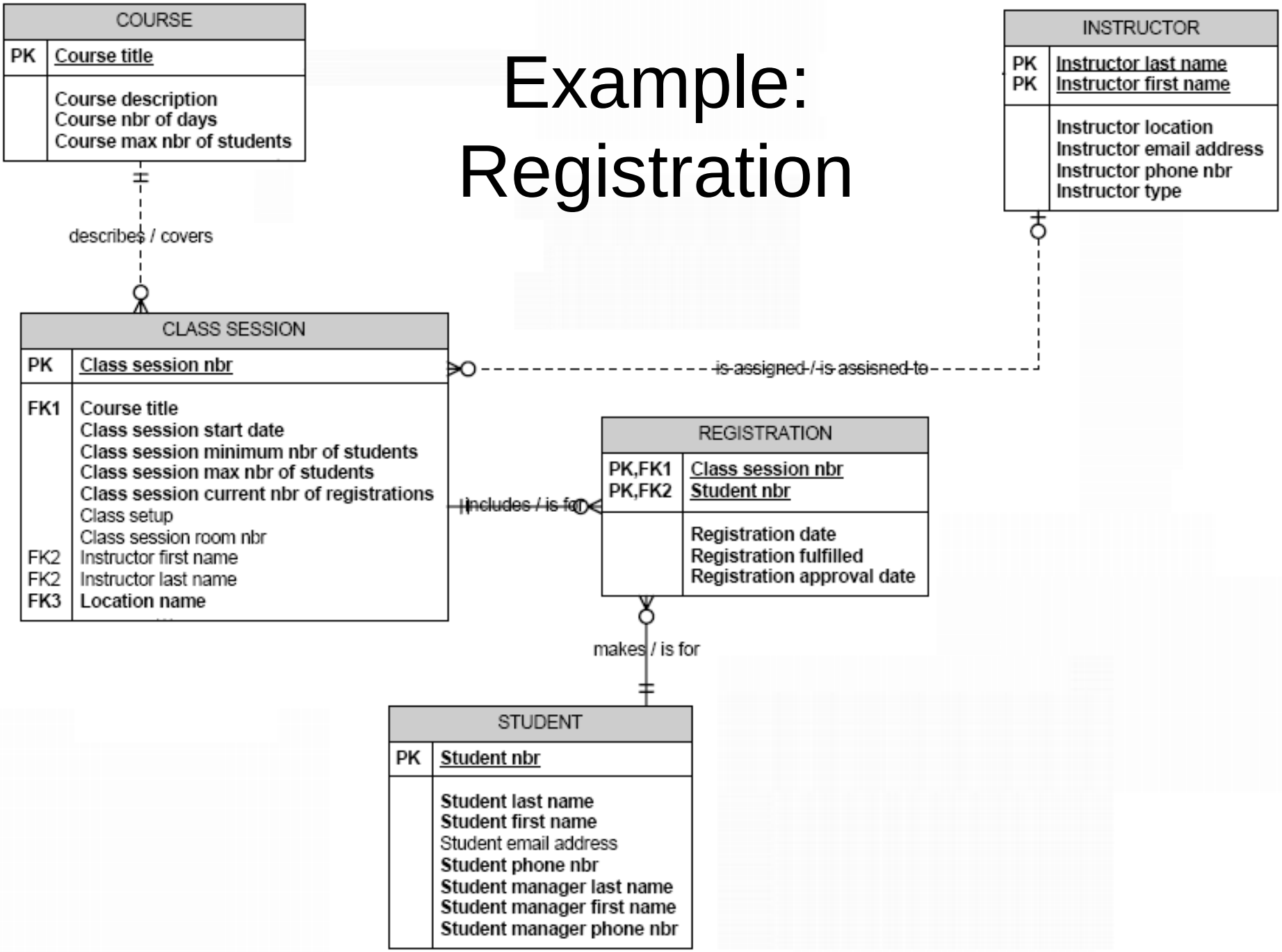
Relationship Symbols



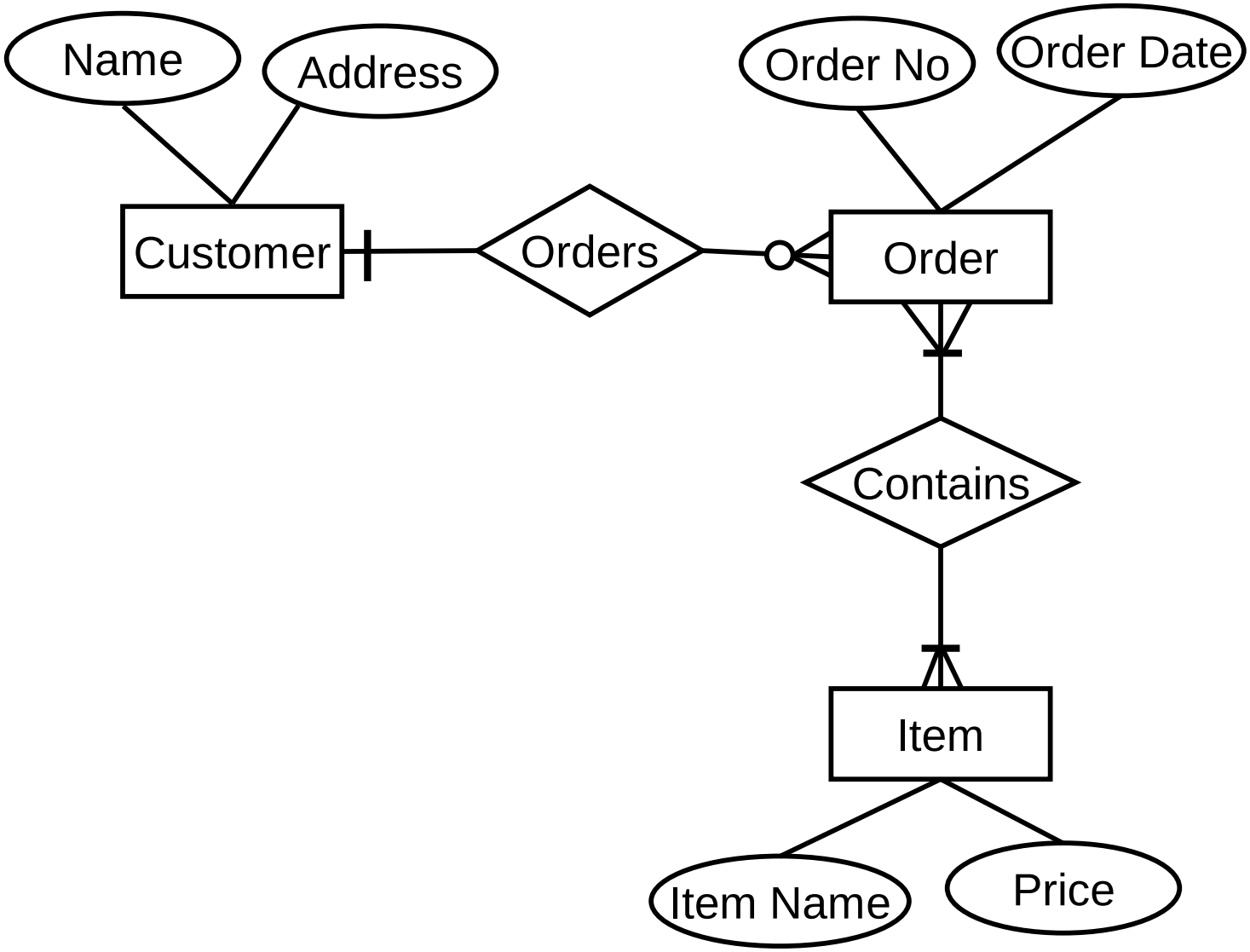
Example: Students and Courses (Bachman notation)



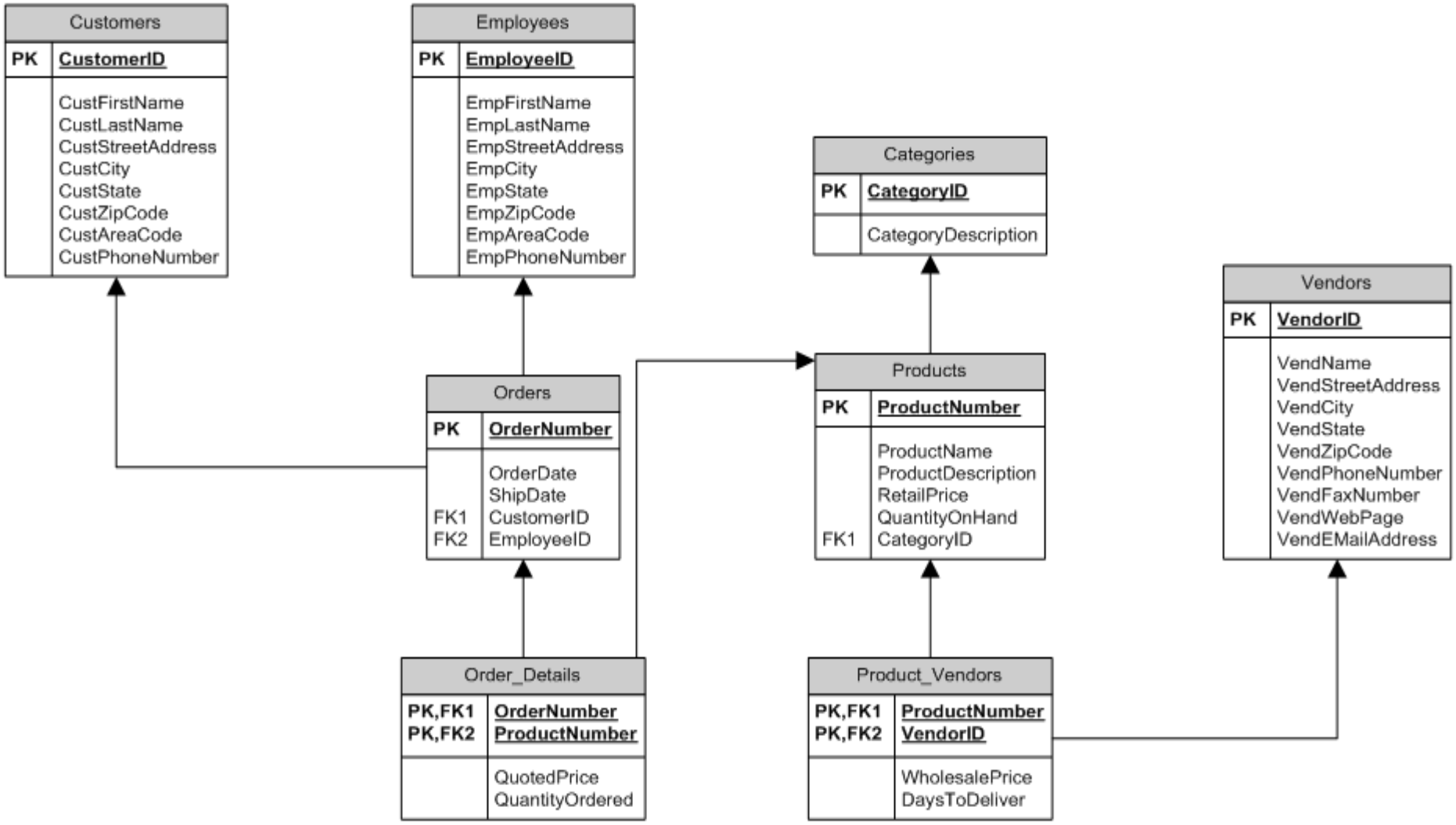
Example: Registration



Example : Orders



Example : Orders and Products



Data Dictionary

- Data dictionary is a collection of data item definitions.
- A data item is described with the followings:

Data Name	the primary name of the composite data item
Aliases	other names for the data item
Where used	data transforms (processes) that use the composite data item
How used	the role of the data item (input, output, temporary storage, etc.)
Description	a notation for representing content
Format	specific information about data types, default values (if known)

Data Dictionary Example

name:	telephone number
aliases:	none
where used/how used:	assess against set-up (output) dial phone (input)

description:

telephone number = [local number | long distance number]

local number = prefix + access number

long distance number = 1 + area code + local number

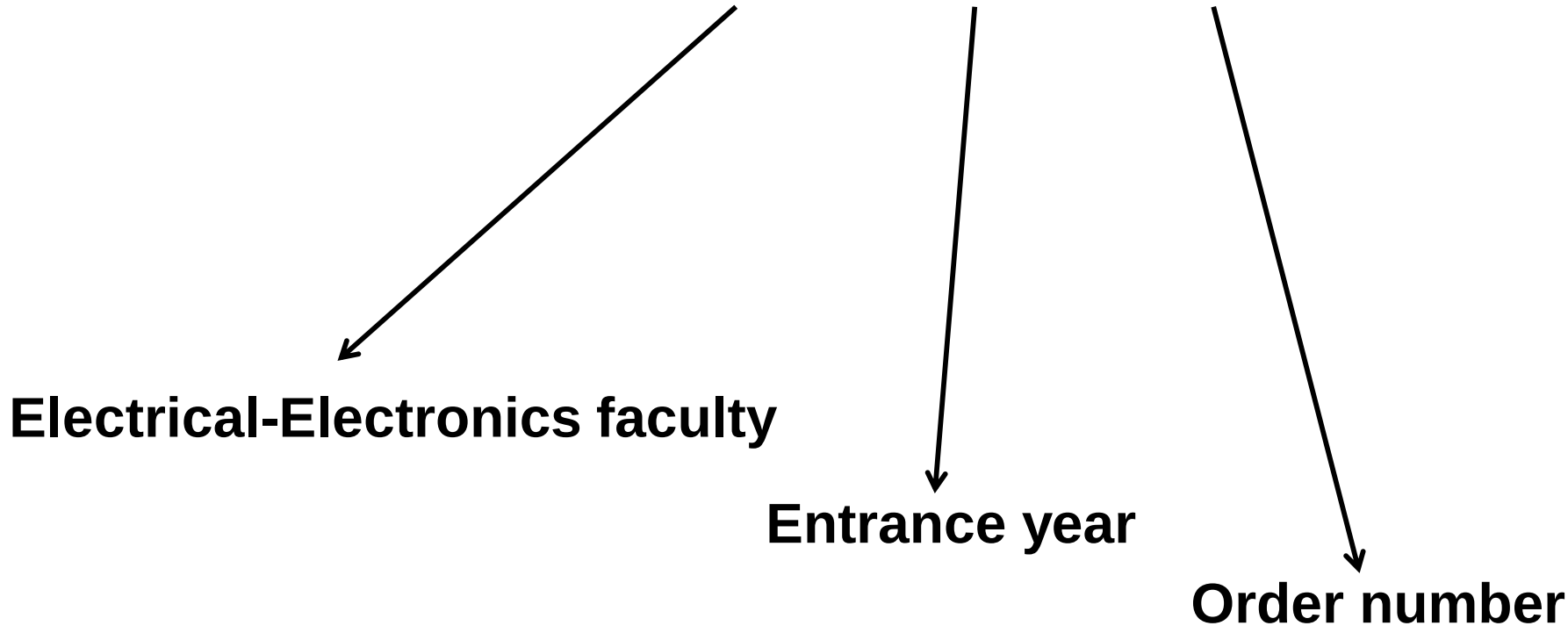
area code = [800 | 888 | 561]

prefix = *a three digit number that never starts with 0 or 1*

access number = * any four number string *

Data Dictionary Example

İTÜ Student ID = 04 – 009 - 0761



1. System Engineering
2. Requirements Engineering
3. Requirements Analysis 

The Functional Model

4.2.2

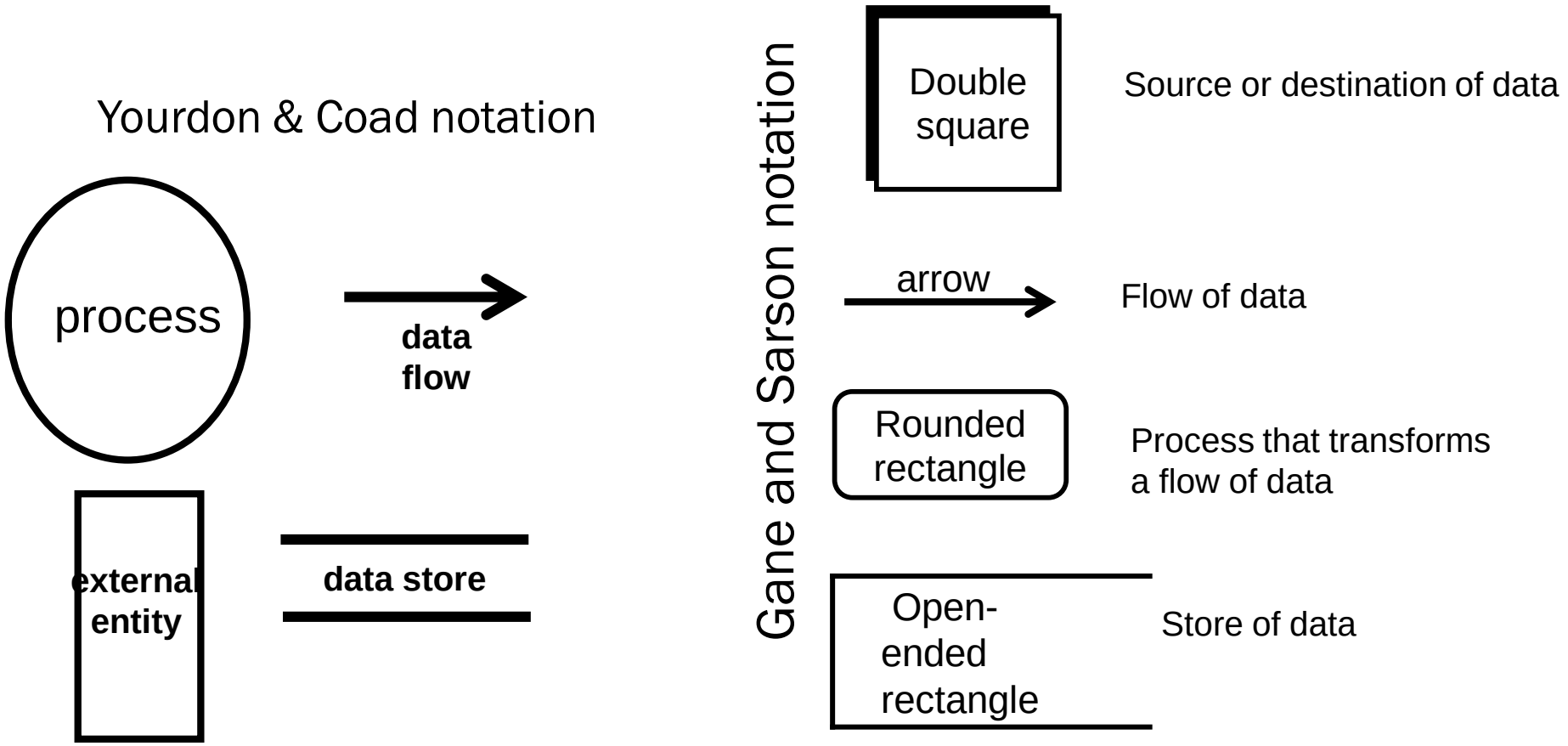
The Functional Model

- Functional modelling is also called Process Modelling.

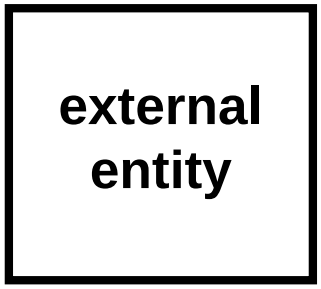
- In functional modelling, Data Flow Diagrams are used.
- There are various DFD notations such as:
  - Yourdon & Coad notation
  - Gane & Sarson notation

- For details of a process one of the followings can be used:
  - Flow Chart
  - Program Description Language (i.e. pseudocode)

Data Flow Diagrams



External Entity



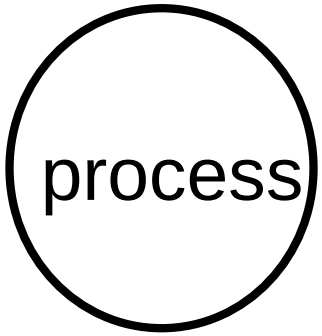
A producer or consumer of data

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

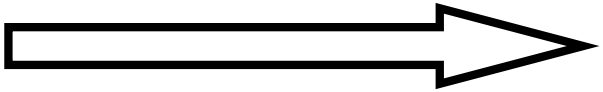


A data transformer (changes input to output)

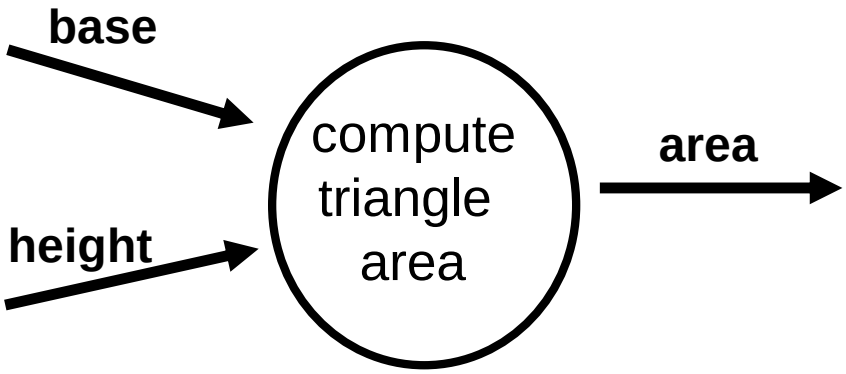
Examples: compute taxes, determine area, format report, display graph

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

Data Flow



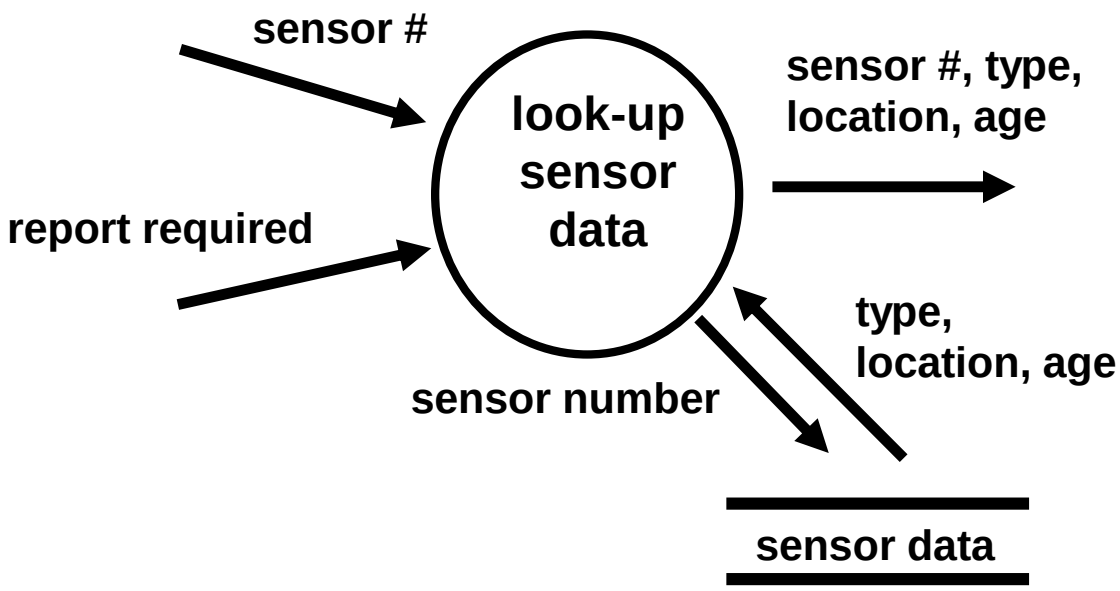
Data flows through a system, beginning as input and be transformed into output.



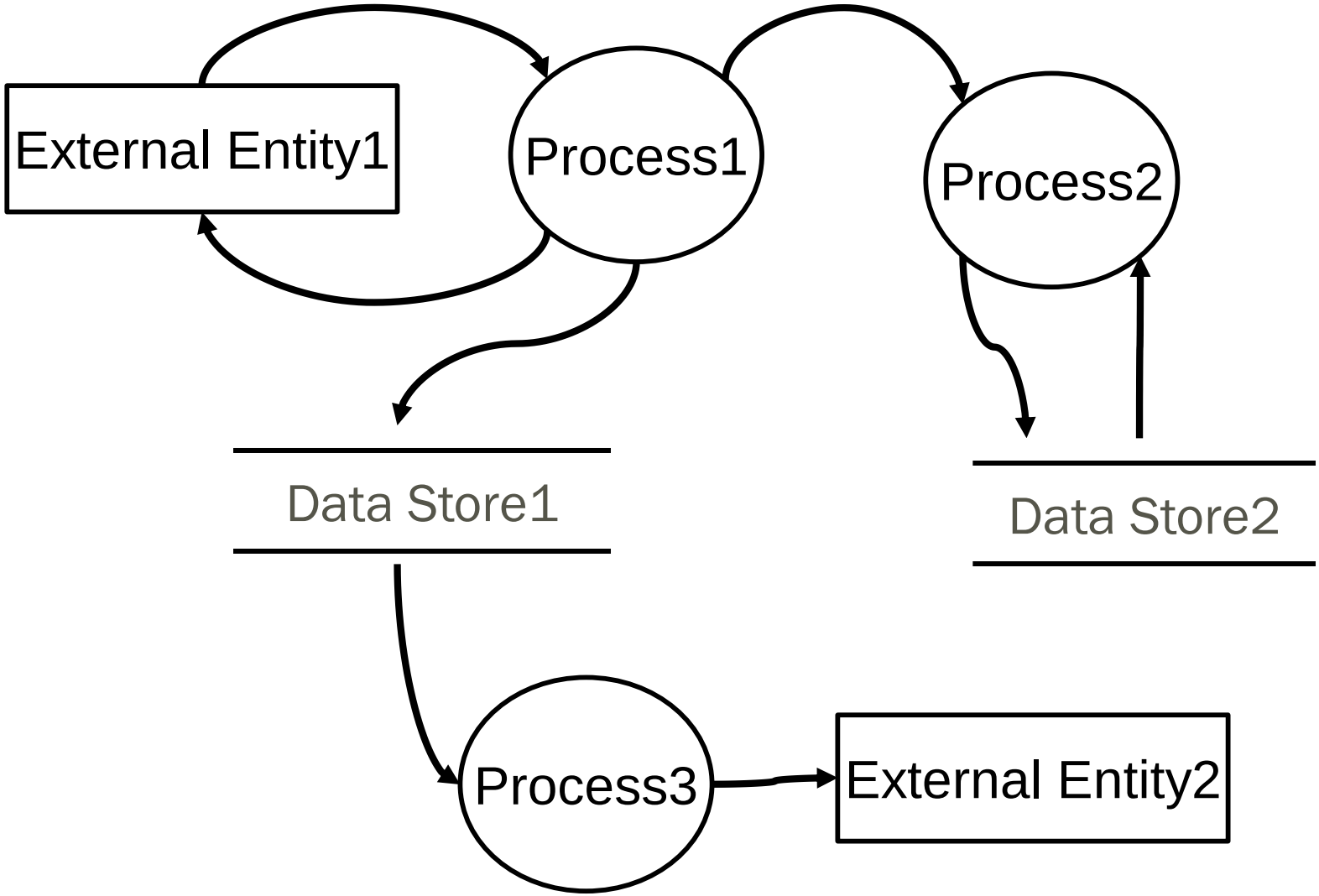
Data Stores

data store

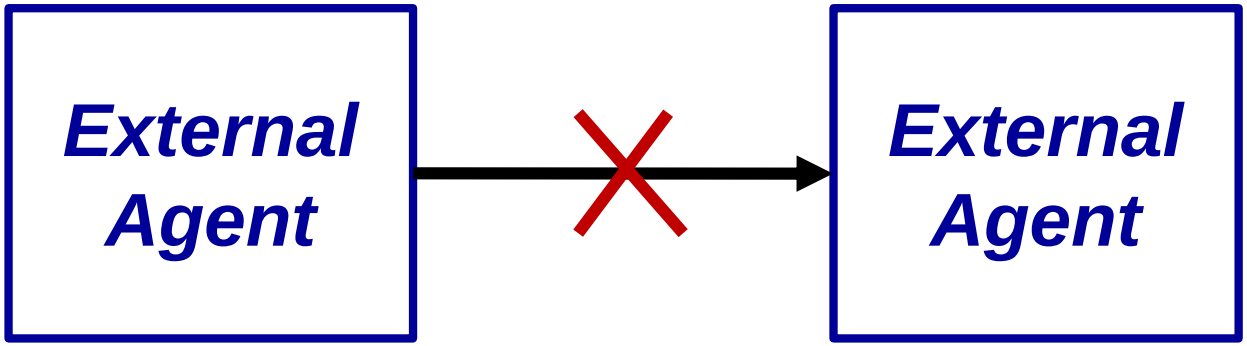
Data is often stored for later use.



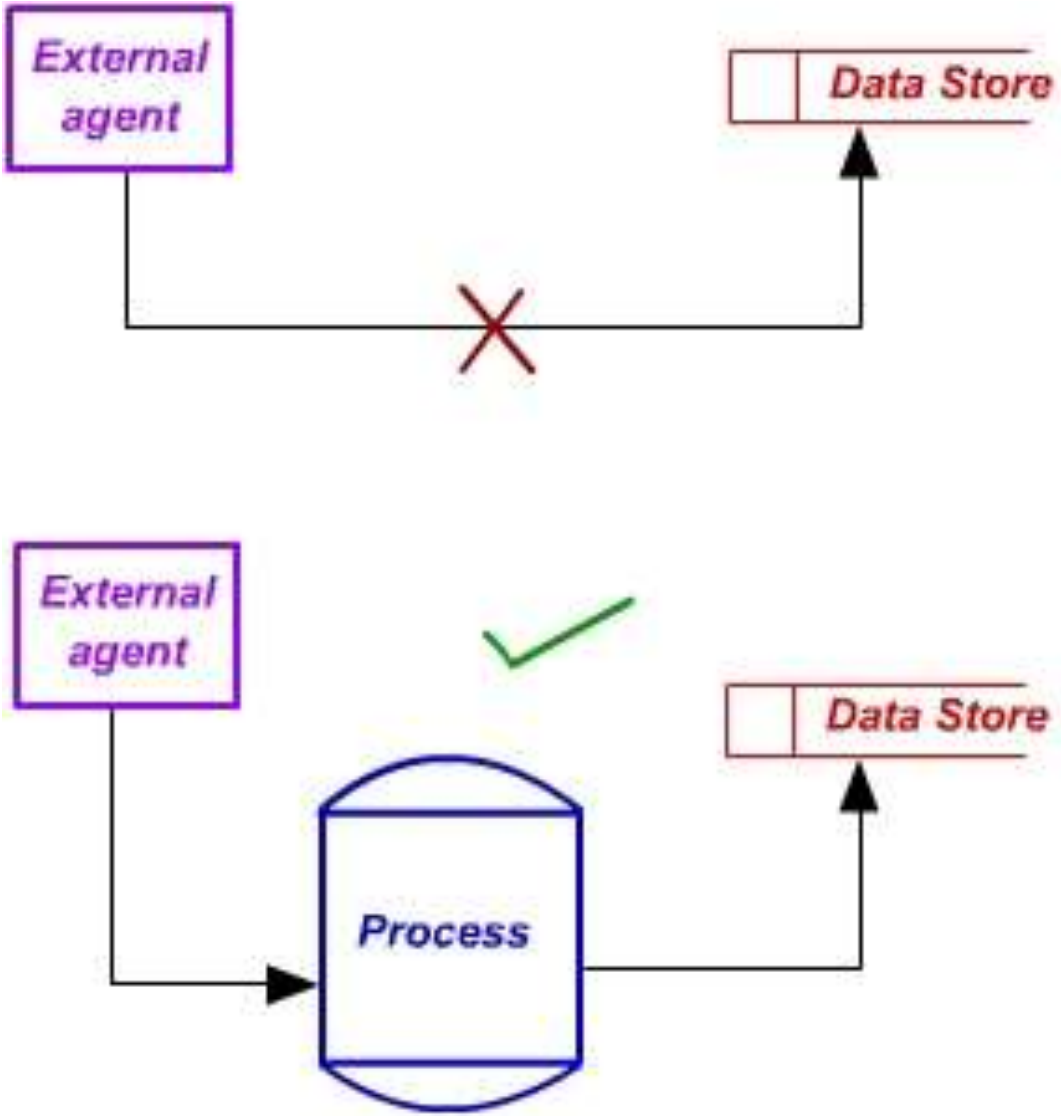
Example: Generic DFD



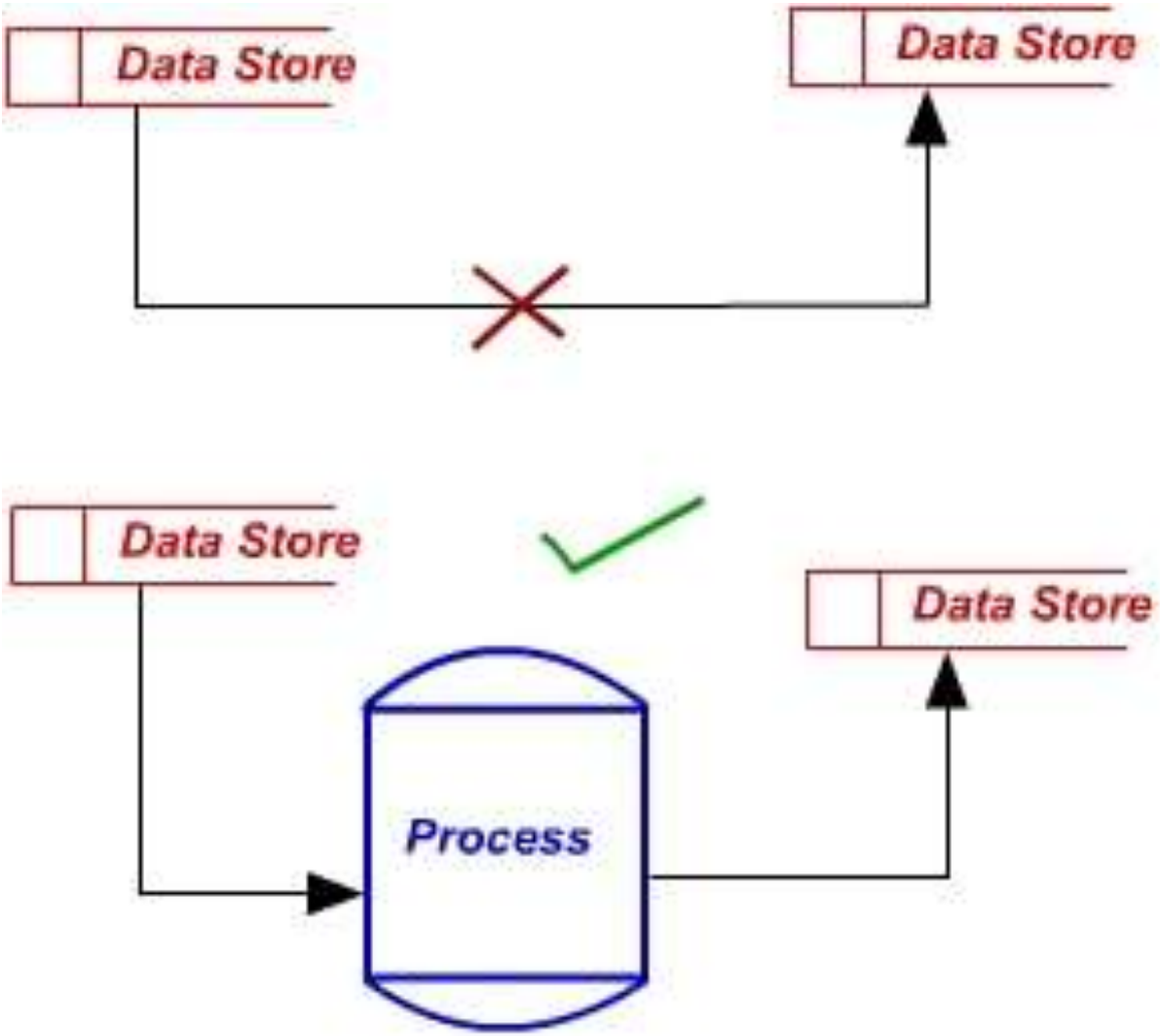
DFD Rules (1)



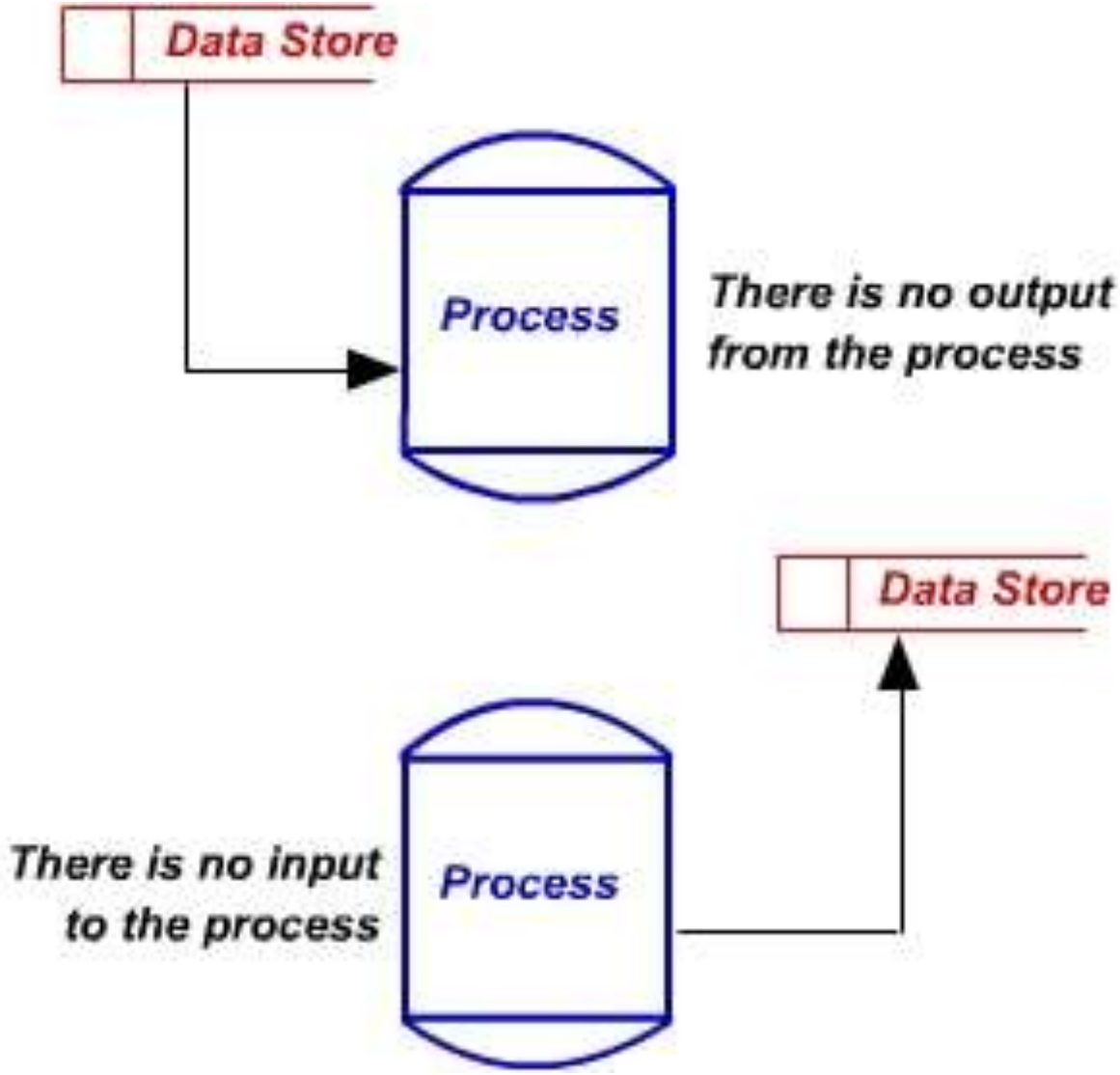
DFD Rules (2)


External agent
Data Store
External agent
Data Store
Process

DFD Rules (3)



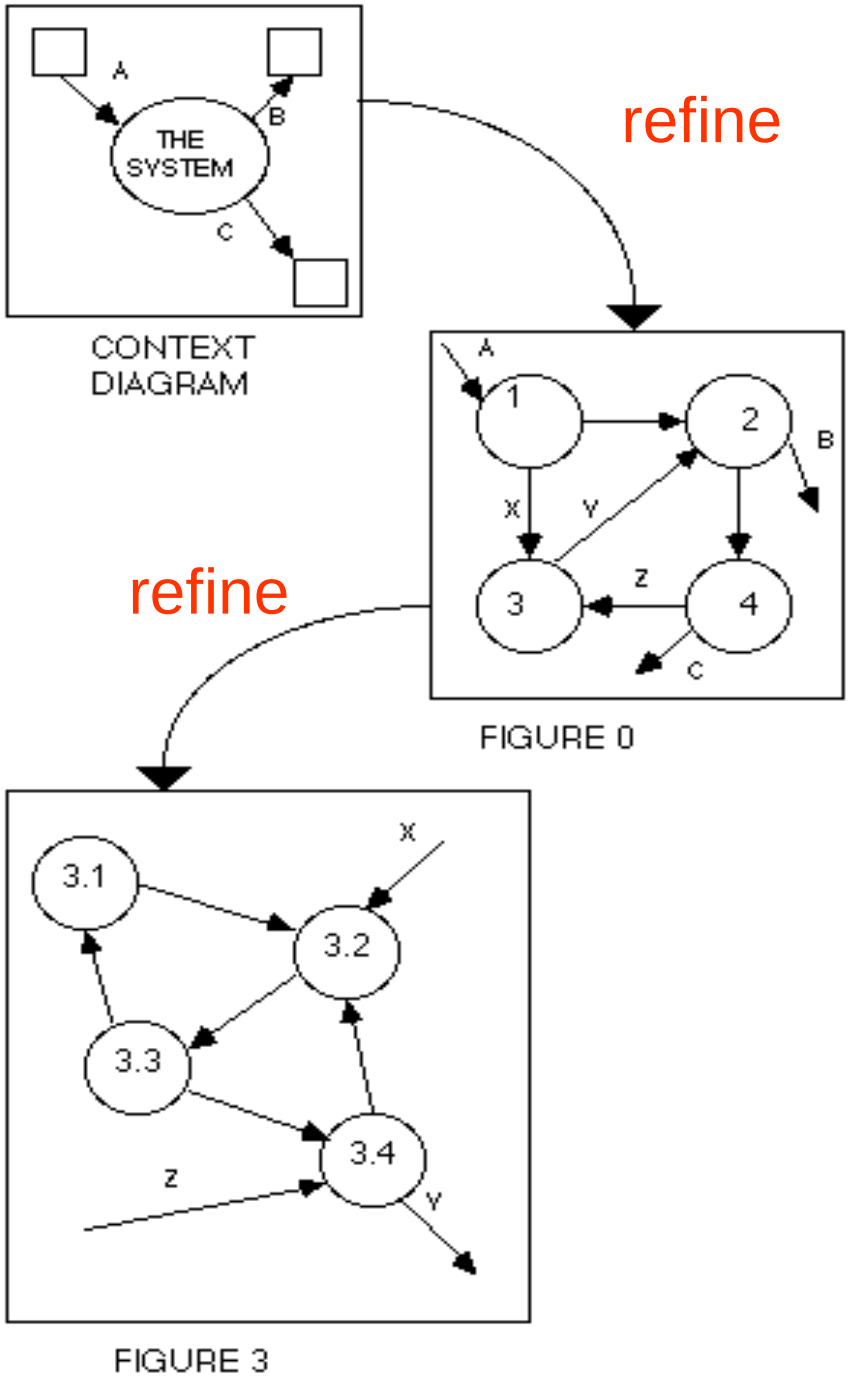
DFD Rules (4)


Data Store
Process
There is no output
from the process
Data Store
There is no input
to the process
Process

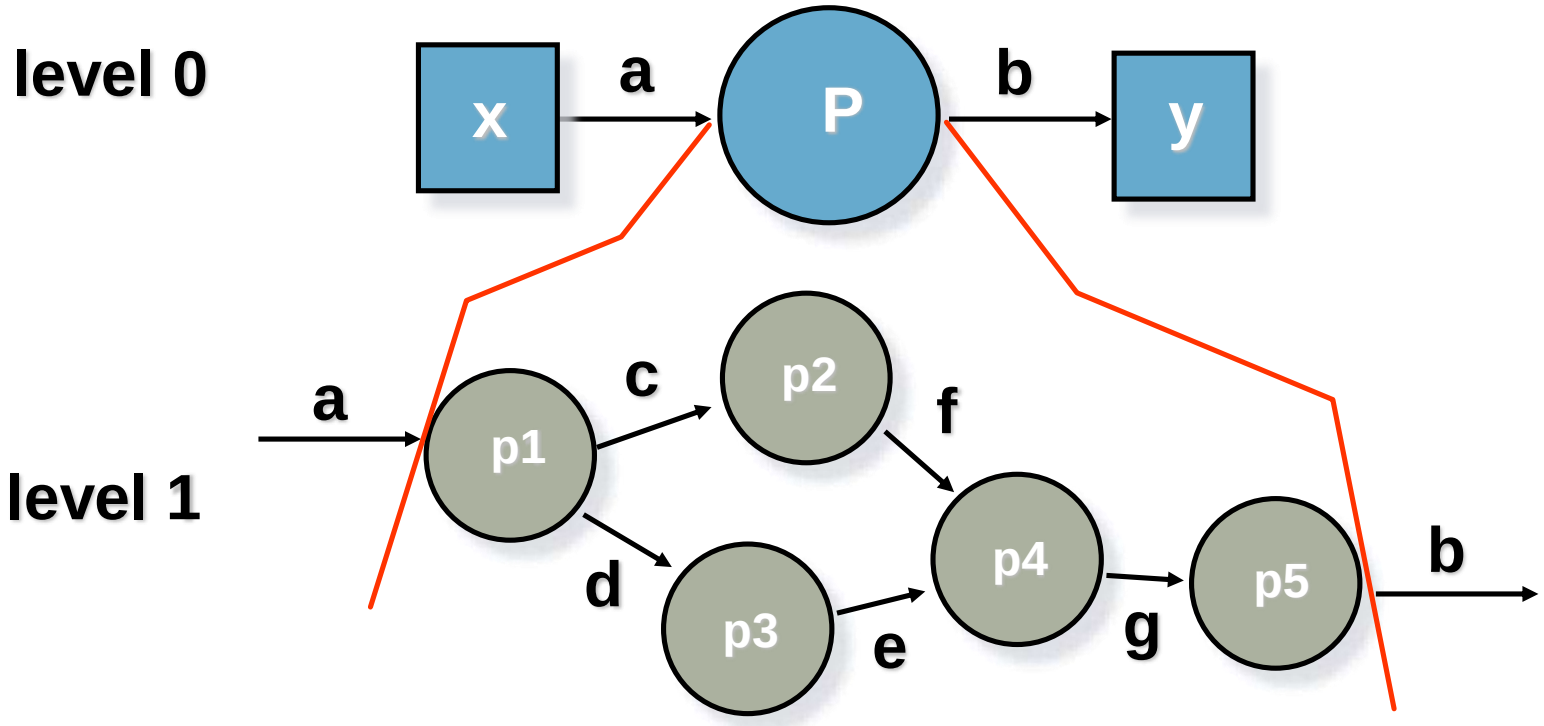
Data Flow Refinement

- DFD modelling is performed from level-0 to level-1, level-2, etc.
- A suggested expansion ratio between one level and the next level is 1:5
- Most systems require between 3 and 7 levels for an adequate flow model
- If a bubble does a number of different things, it needs further refinement.
- Each bubble is refined until it does just one thing
- The expansion ratio decreases as the number of levels increase

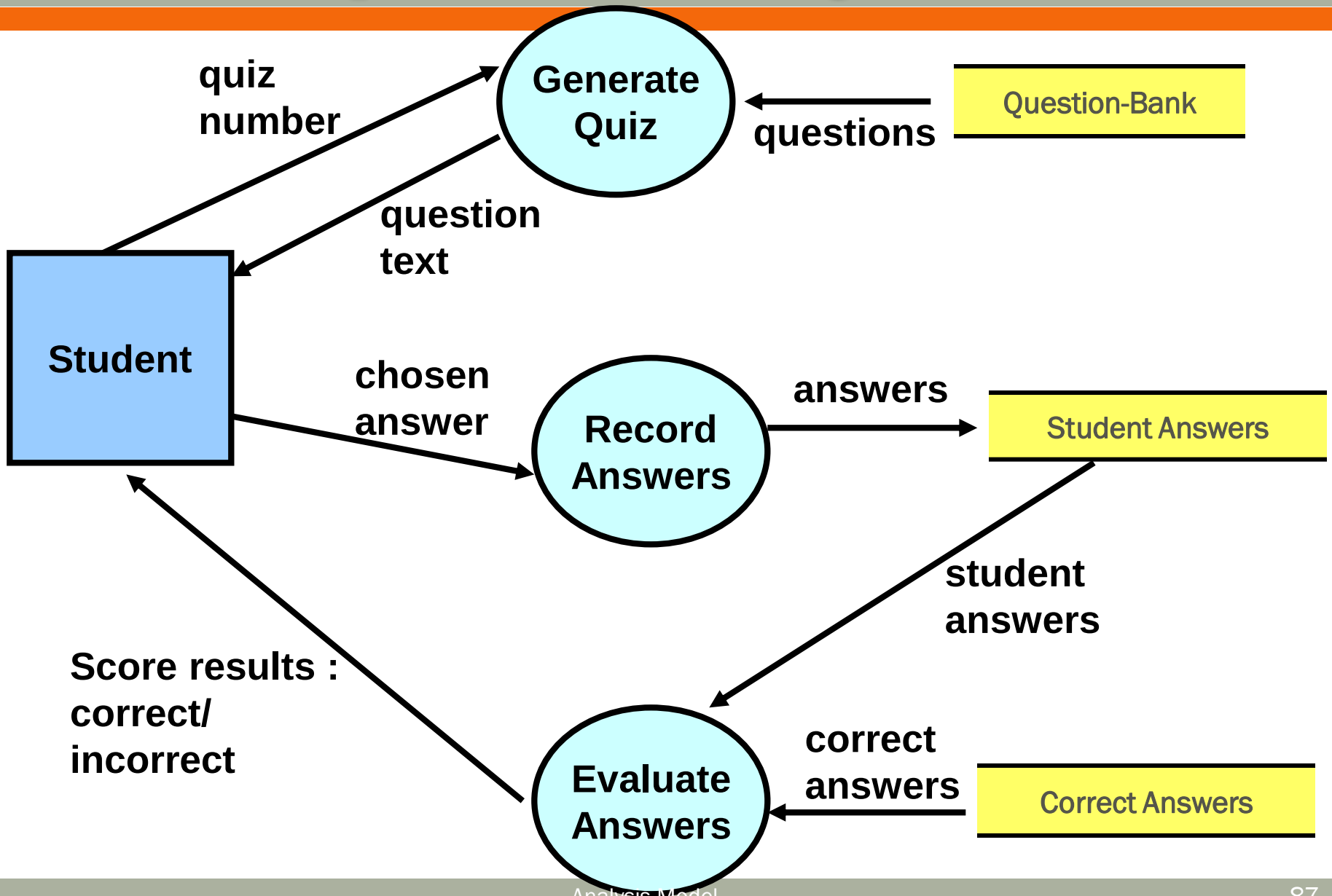
Example of Data Flow Refinement



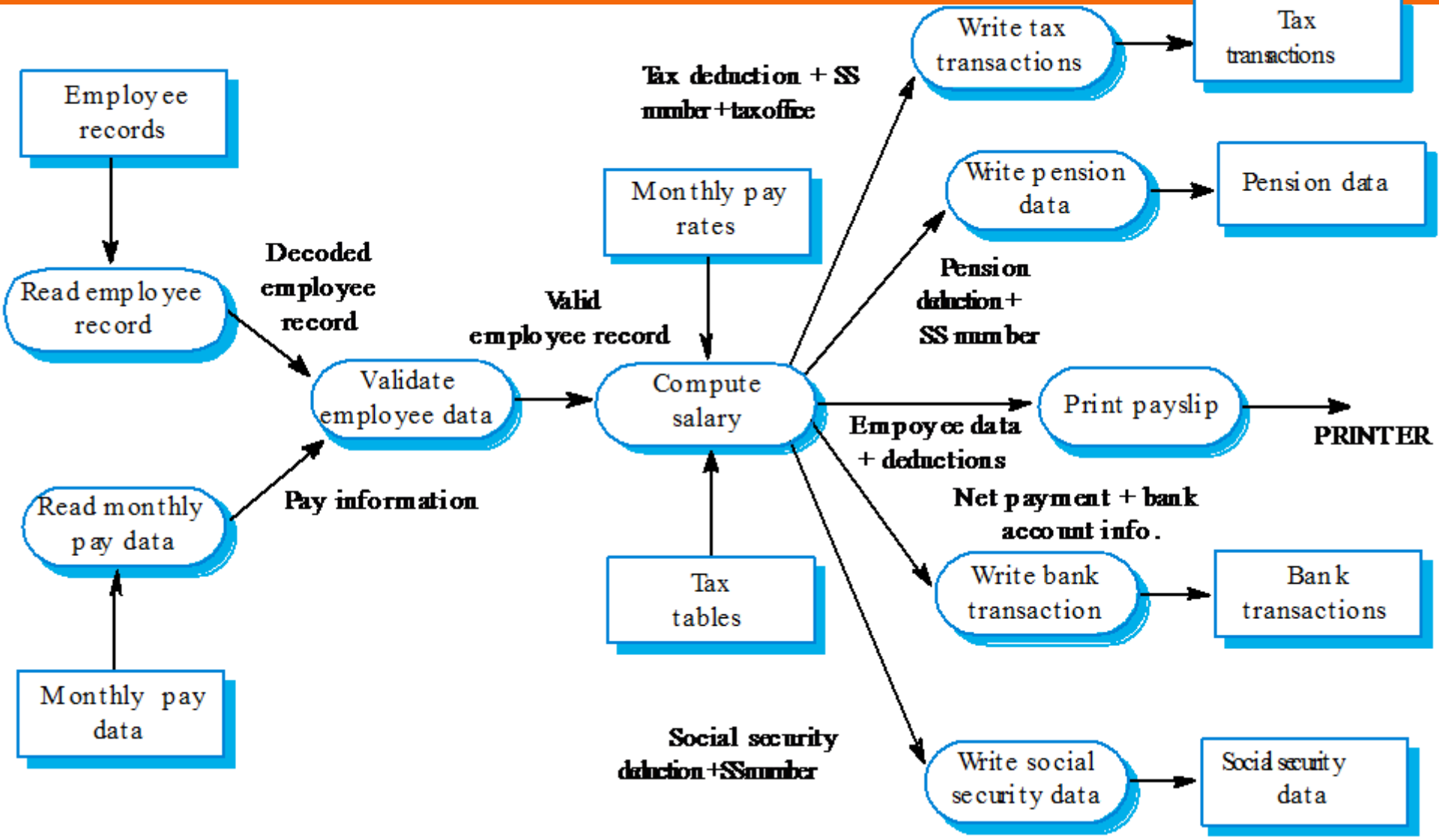
Example Data Flow Hierarchy



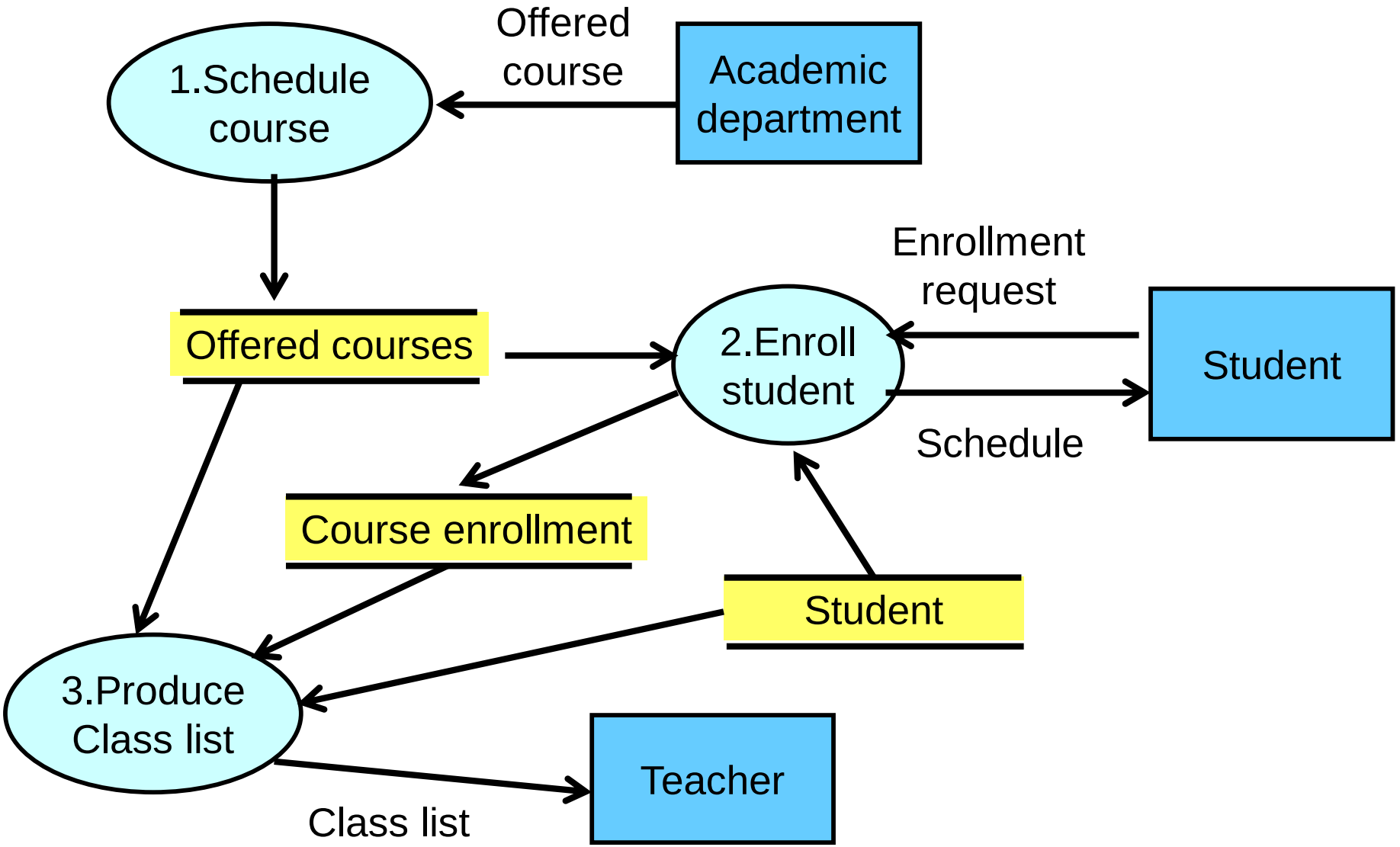
Example: DFD for Quizzing Software



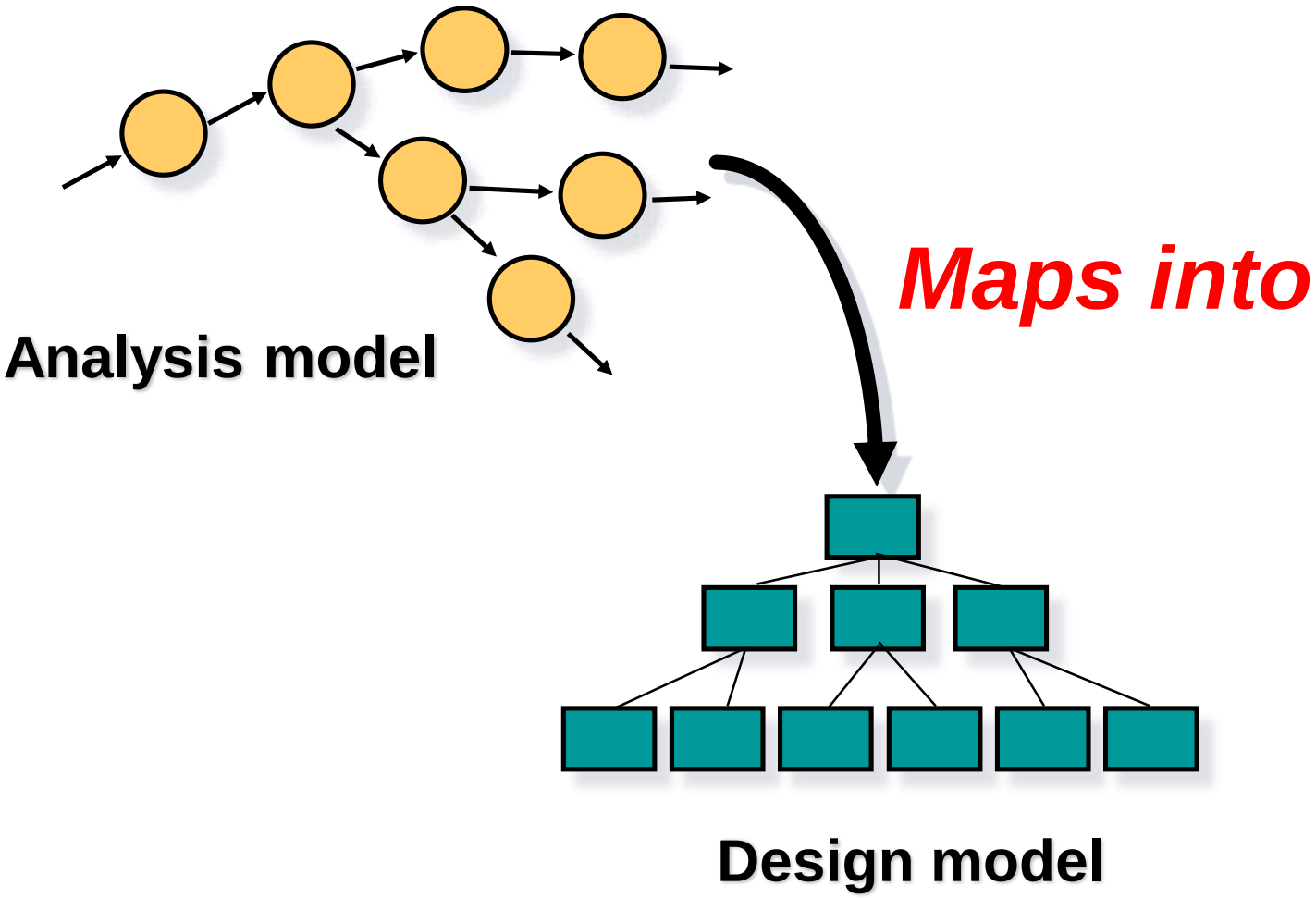
Example: DFD for Employees



Example: DFD for Courses



DFDs: A Look Ahead



1. Requirements Engineering
2. Requirements Analysis

The Behavioural Model

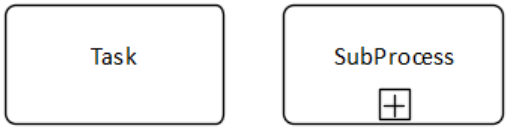
4.2.3

Behavioral Modeling

- BPMN, YAWL, Activity Diagrams (Swimlanes)
- State machines (statecharts), Petri-nets
- Pseudocodes and Structured texts
- Control Flow Diagrams, Flowcharts

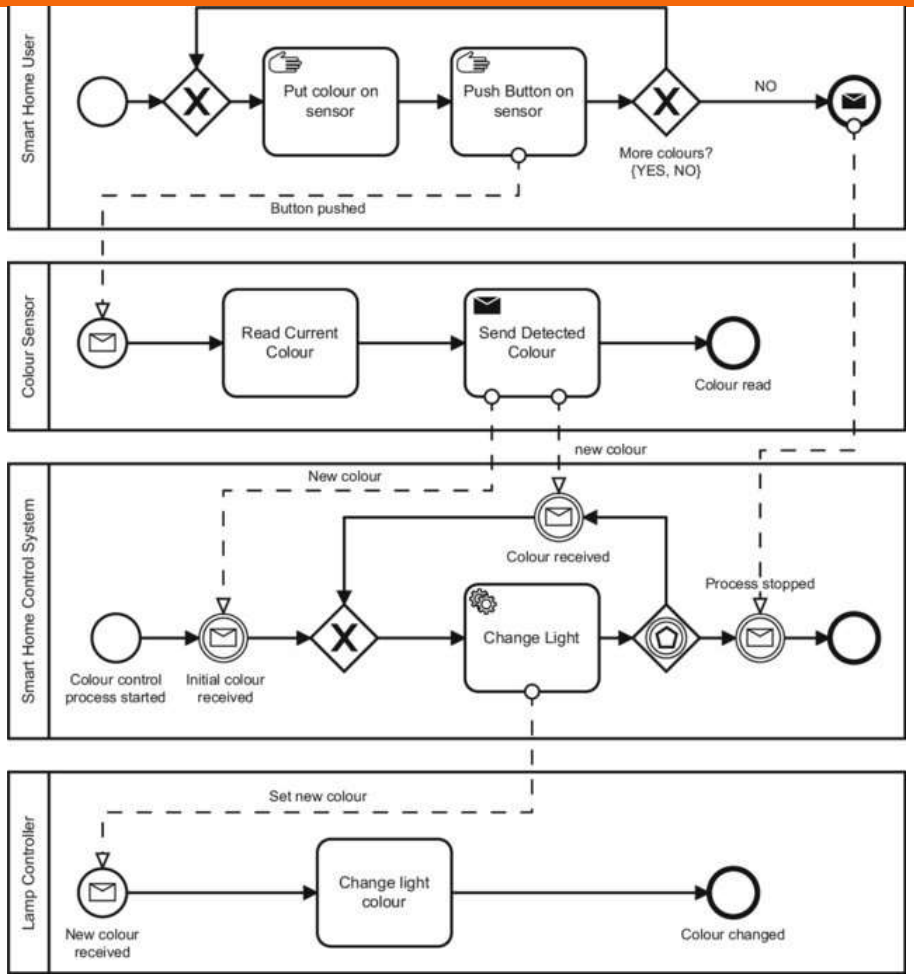
BUSINESS PROCESS MODELING

Activities

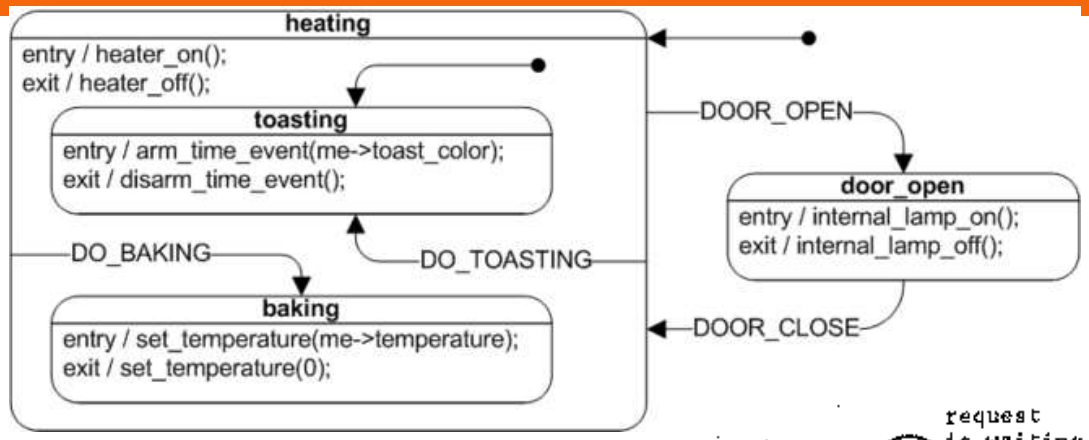


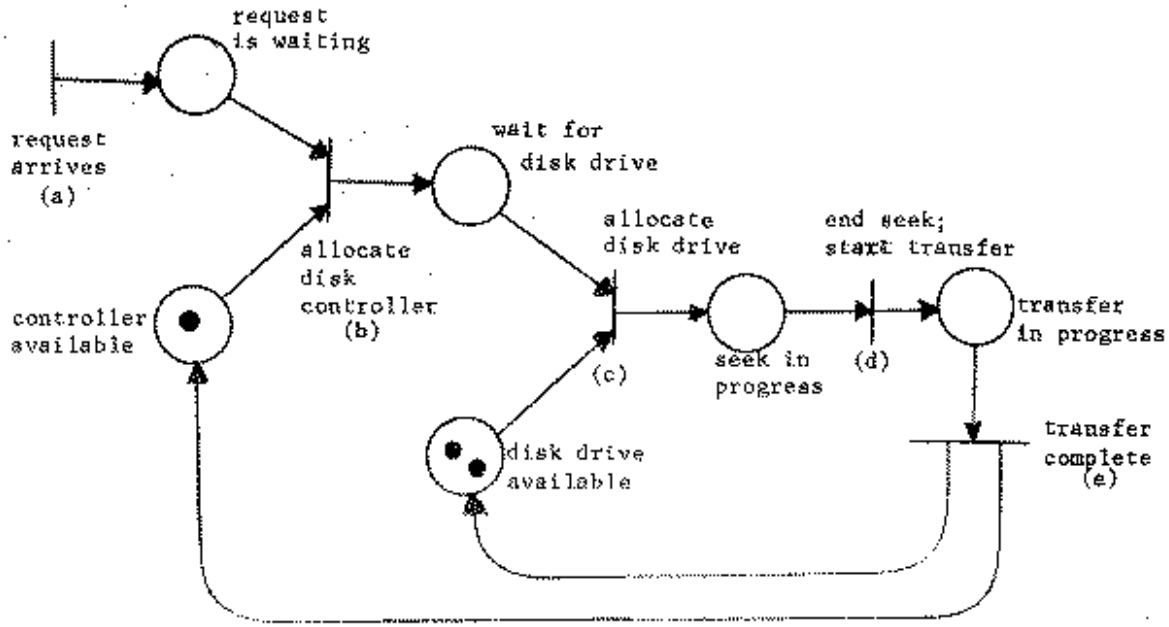
Events

Gateways



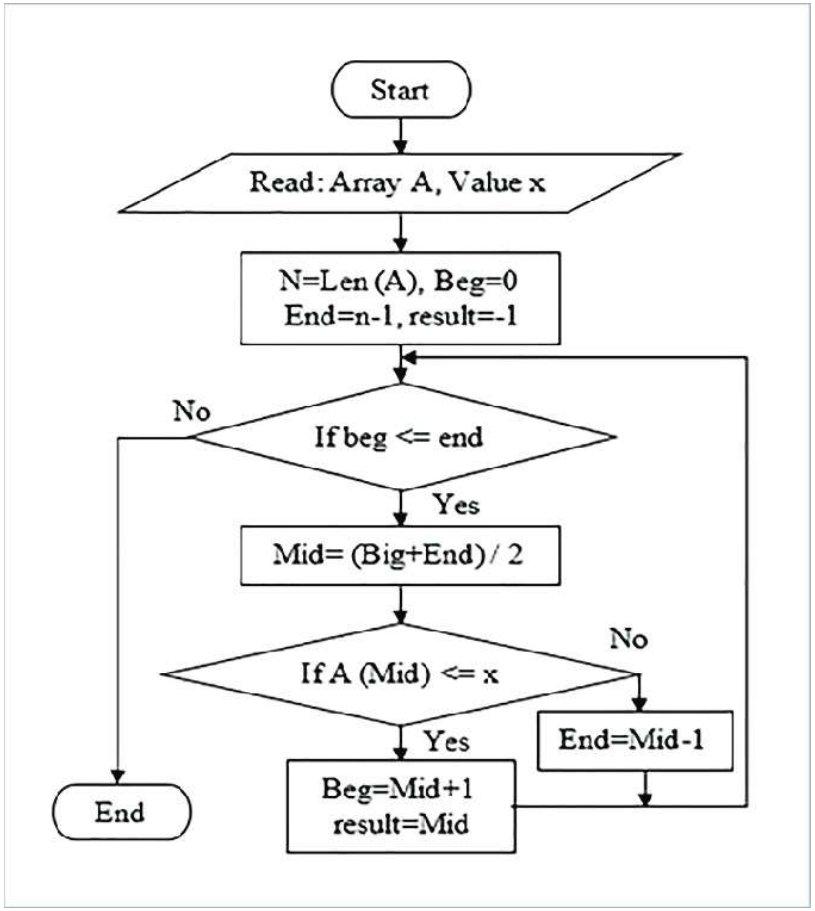
STATE MACHINE – PETRI NET





FLOWCHART

```
function binary_search(A, n, T) is
    L := 0
    R := n - 1
    while L ≤ R do
        m := floor((L + R) / 2)
        if A[m] < T then
            L := m + 1
        else if A[m] > T then
            R := m - 1
        else:
            return m
    return unsuccessful
```



The Behavioural Model

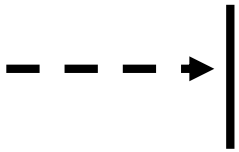
- In behavioural modelling, Control Flow Diagrams and State Transition Diagrams are used.
- Control Flow Diagrams is mostly used in Embedded or Real-time software development.
- The control flow diagram is superimposed on the DFD and shows events that control the processes noted in the DFD.
- Control flows (events and control items) are noted by dashed arrows.

Control Flow Diagrams

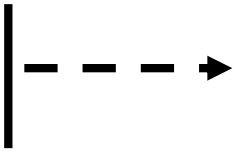
- Represents “events” and the processes that manage events
- An “event” is a Boolean condition that can be ascertained by:
  - listing all sensors that are "read" by the software.
  - listing all interrupt conditions.
  - listing all "switches" that are actuated by an operator.
  - listing all data conditions.
  - Examining the processing narrative, review all "control items" as possible CSPEC inputs/outputs.
  - A CSPEC is shown with a **State Transition Diagram**.

Control Flow Diagrams

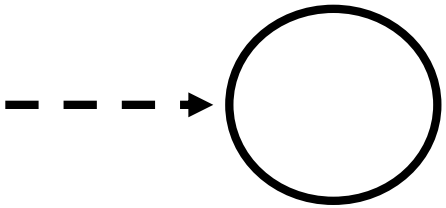
- a dashed arrow entering a vertical bar is an input



- a dashed arrow leaving a process implies a data condition



- a dashed arrow entering a process implies a control input read directly by the process



Example:

Vending Machines
Management Software

Statement of Software Scope (1)

- You've been asked to develop a management software for a company which maintains a large number of vending machines (self-service machines to sell snack foods).
- Vending machines are at several locations across the city.
- Each location can have one or more machines.
- Vending machines need to be refilled with different quantities depending on the consumption at each location.

Statement of Software Scope (2)

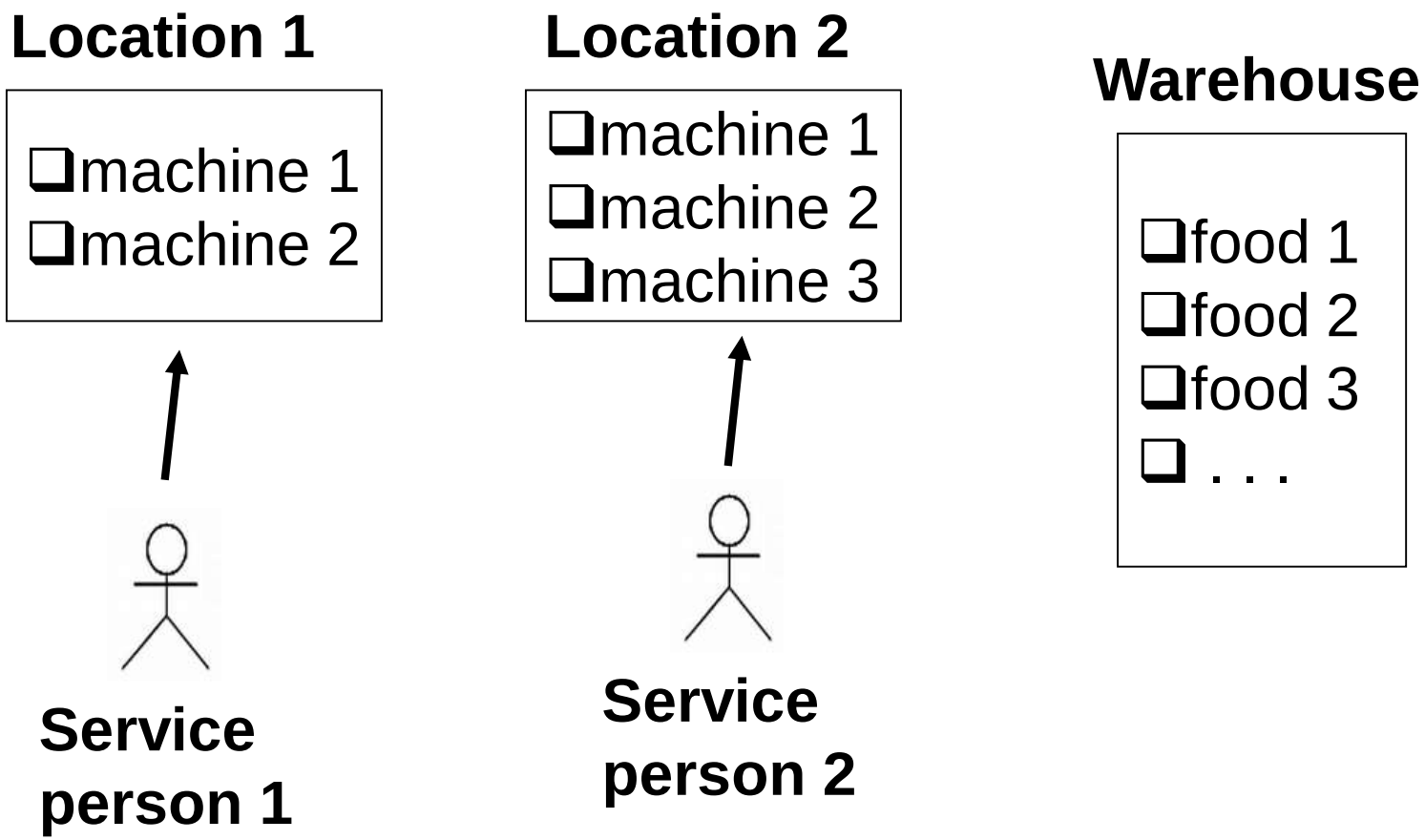
- Each location is served by one service personnel.
- All foods are stored in the company's warehouse.
- Before a personnel leaves for servicing, he requests foods from the warehouse for refilling.
- After returning from servicing, the service personnel submits the cash he collected from each machine to the company; returns any unused foods; and informs the company of any problems with the machines.
- When the food stock gets low, a purchase order is generated.

Statement of Software Scope (3)

- The company wants to manage their business using the software that keeps track of the
  - locations,
  - machines,
  - service personnel,
  - food stocks,
  - maintenance history for machines,
  - the amount of food requested and returned by each personnel,
  - total cash generated per machine, per location,
  - details of any purchase orders generated.

- Daily reports (such as total cash report, maintenance summary report, purchase order) will need to be generated.

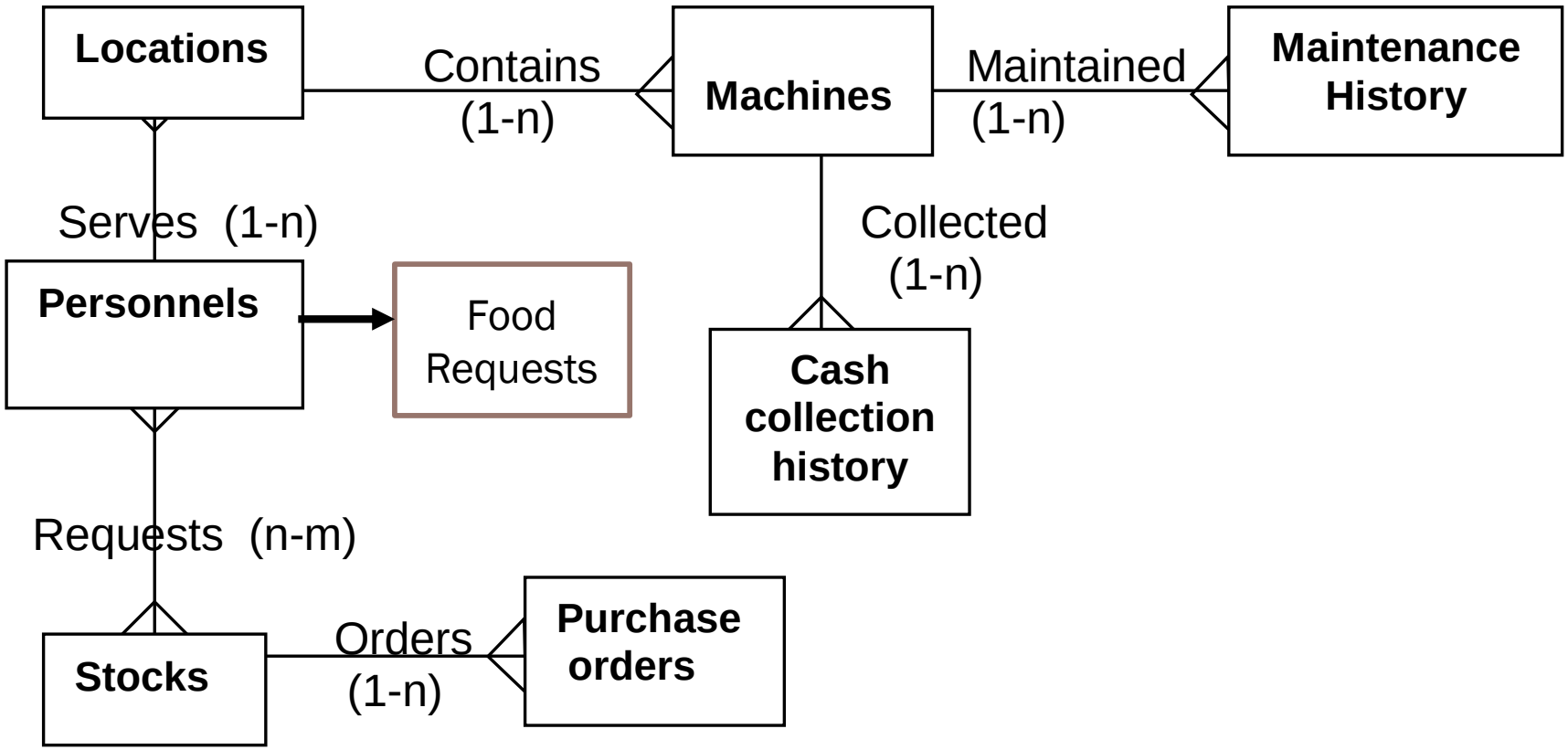
System Outline



Tasks

- Draw an Entity Relationship Diagram that describes the relationships between the different data entities.
  - For each relationship, name the relationship and define its cardinality (1-1, 1-n, or n-m).
  - For each entity, list all data items.
- Produce Level-0 and Level-1 Data Flow Diagrams that captures the main processes, data flows, information sources and data stores of this application.
- Produce a Program Structure Chart.

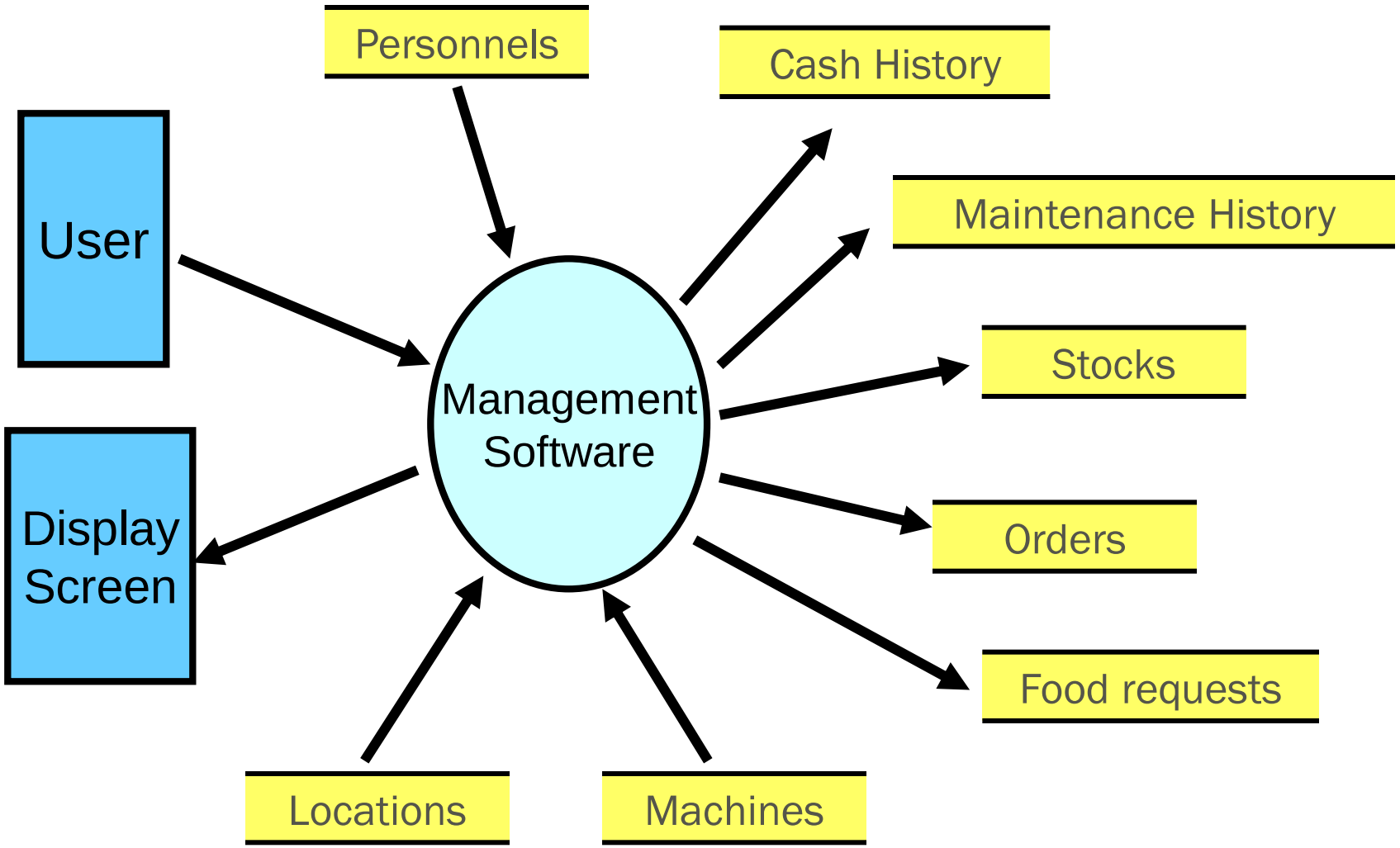
Entity Relationship Diagram (ERD)



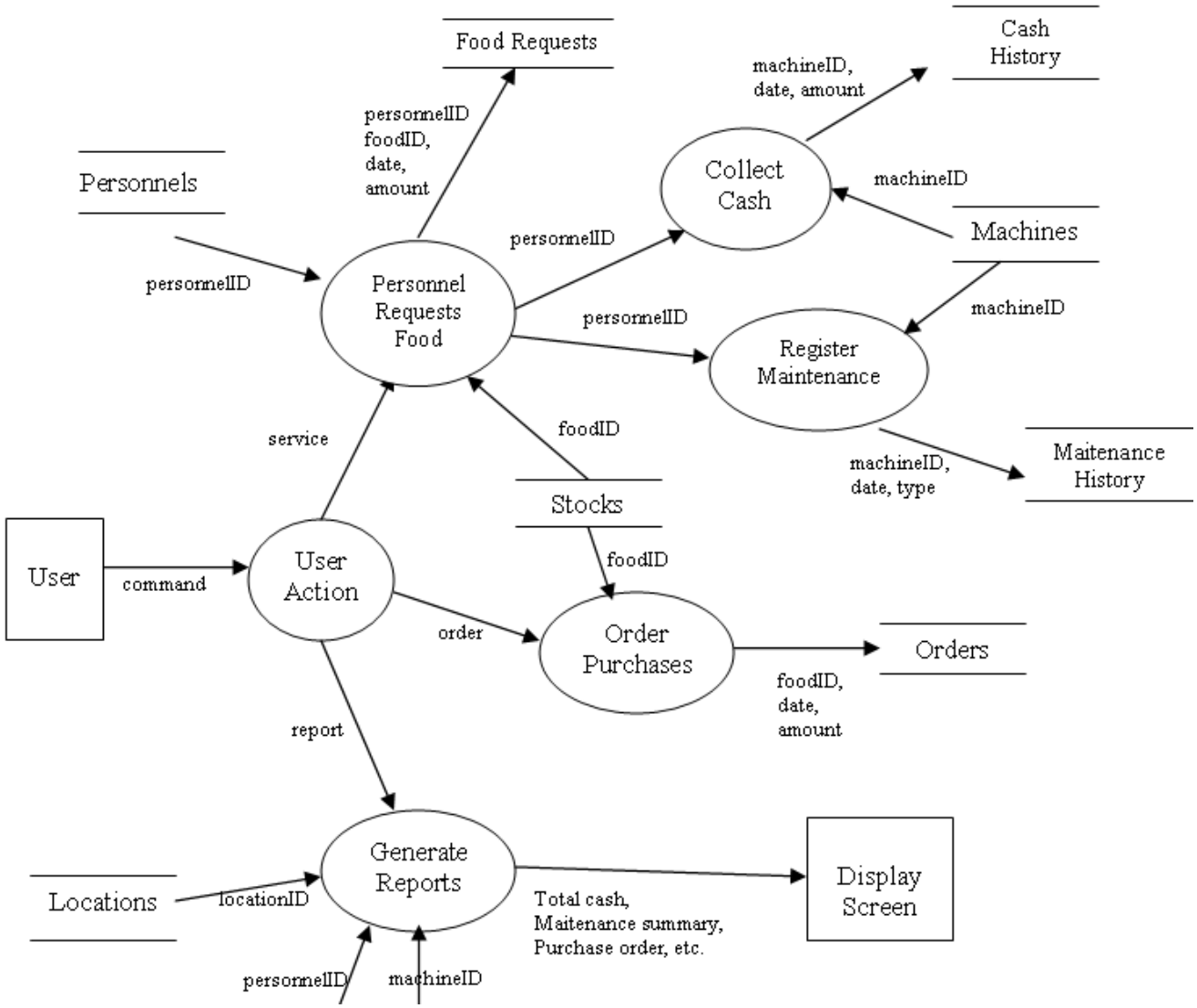
Entities

ENTITIY	DATA ITEMS
Locations	Location_ID, Address, Number of consumers, ServicePersonnel_ID
Machines	Machine_ID, Location_ID, Frequency of refilling
Personnels	Personnel_ID, Personnel name
Stocks	Food_ID, Food name, Current amount
Food_Requests	Personnel_ID, Food_ID, Date of request, Amount of request, Returned amount
Cash_Collection_History	Machine_ID, Date of collection, Amount of cash
Maitenance_History	Machine_ID, Date of maintenance, Type of maintenance
Purchase_Orders	Order_ID, Food_ID, Date of order, Amount of order

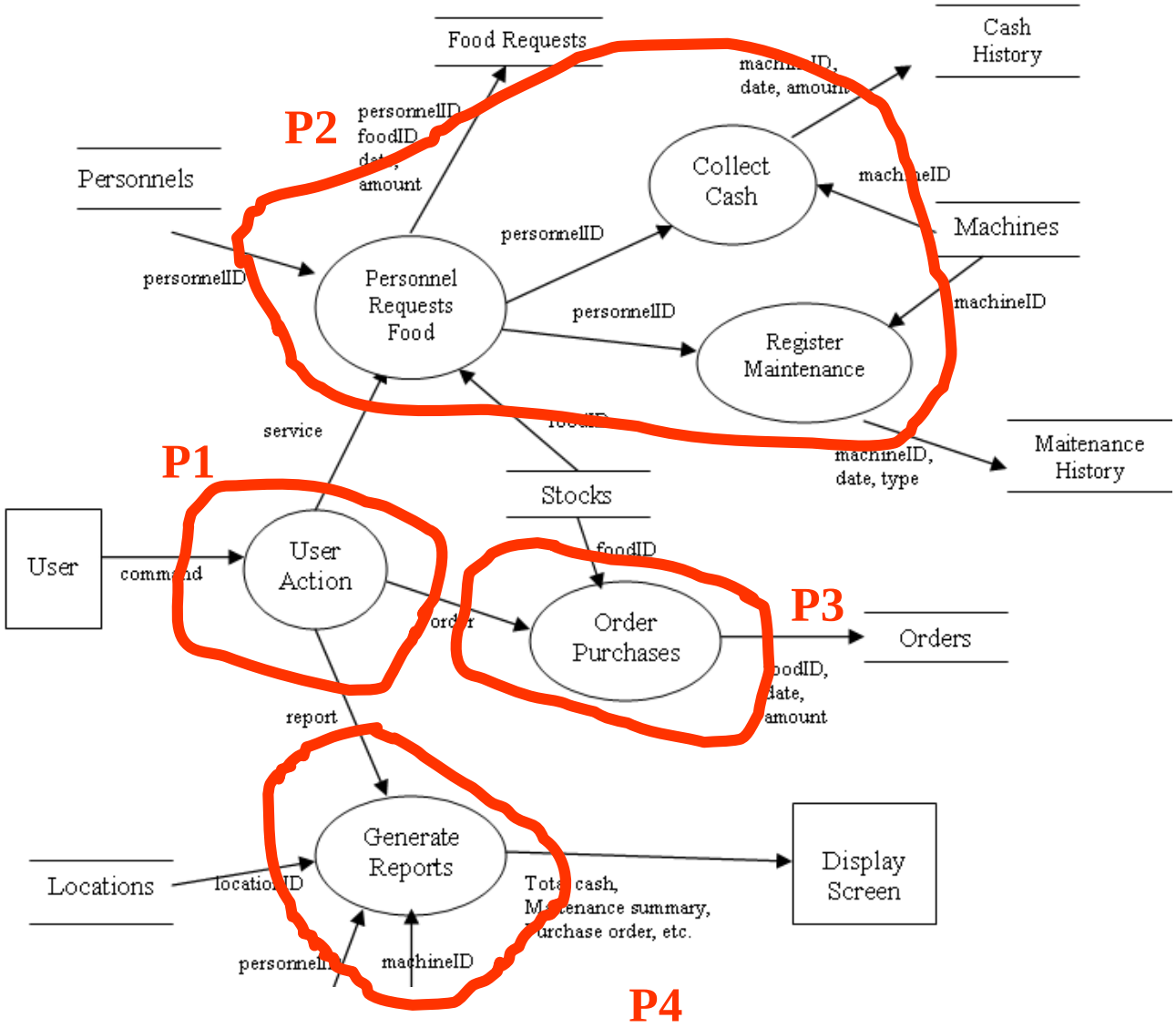
Level-0 DFD


Personnels
Cash History
User
Maintenance History
Management Software
Stocks
Display Screen
Orders
Food requests
Locations
Machines

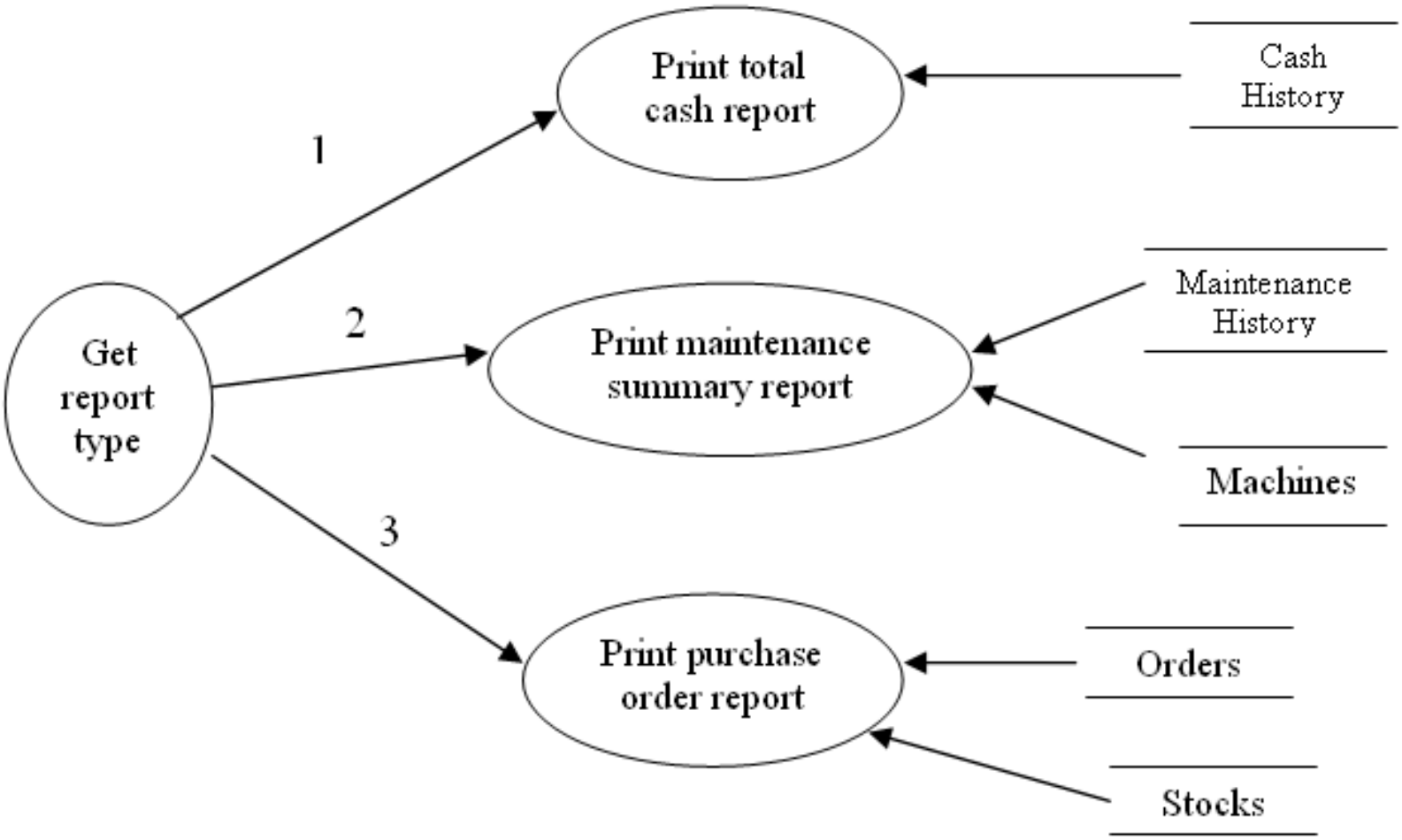
Level-1 DFD


Food Requests
Cash History
machineID, date, amount
personnelID foodID, date, amount
Personnels
Collect Cash
machineID
Machines
personnelID
personnelID
Personnel Requests Food
personnelID
machineID
Register Maintenance
service
foodID
machineID, date, type
Maitenance History
Stocks
User
command
User Action
foodID
order
Order Purchases
Orders
foodID, date, amount
report
Generate Reports
Locations
locationID
Total cash, Maitenance summary, Purchase order, etc.
Display Screen
personnelID
machineID

Isolating the flows in Level-1 DFD



Level-2 DFD for P4



Example:

Technical Service
Management Software

Statement of Software Scope (1)

- A technical service firm needs a web-based software to keep track of maintenance and repairment operations for their customers' devices such as combi, air conditioner, laundry machine, refrigerator, etc.

- The followings are functional requirements:

- A "Service Request Form" must be filled for all kinds of service requests. The form must contain fields for *customer name, address, telephone, and a description of service* being requested.

Statement of Software Scope (2)

- The request will be tracked by a status code:
  - "Device will be picked up from customer"
  - "Device will be serviced at customer's place"
  - "Device is in service"
  - "Device waiting for delivery to customer"
  - "Delivery completed"

- A request can be done directly by a customer over the Internet, or an authorized personnel can record the request for the customer.

- Customer should be able to query the status of his service request.

Statement of Software Scope (3)

- The manager will assign a service request task to an available technician.

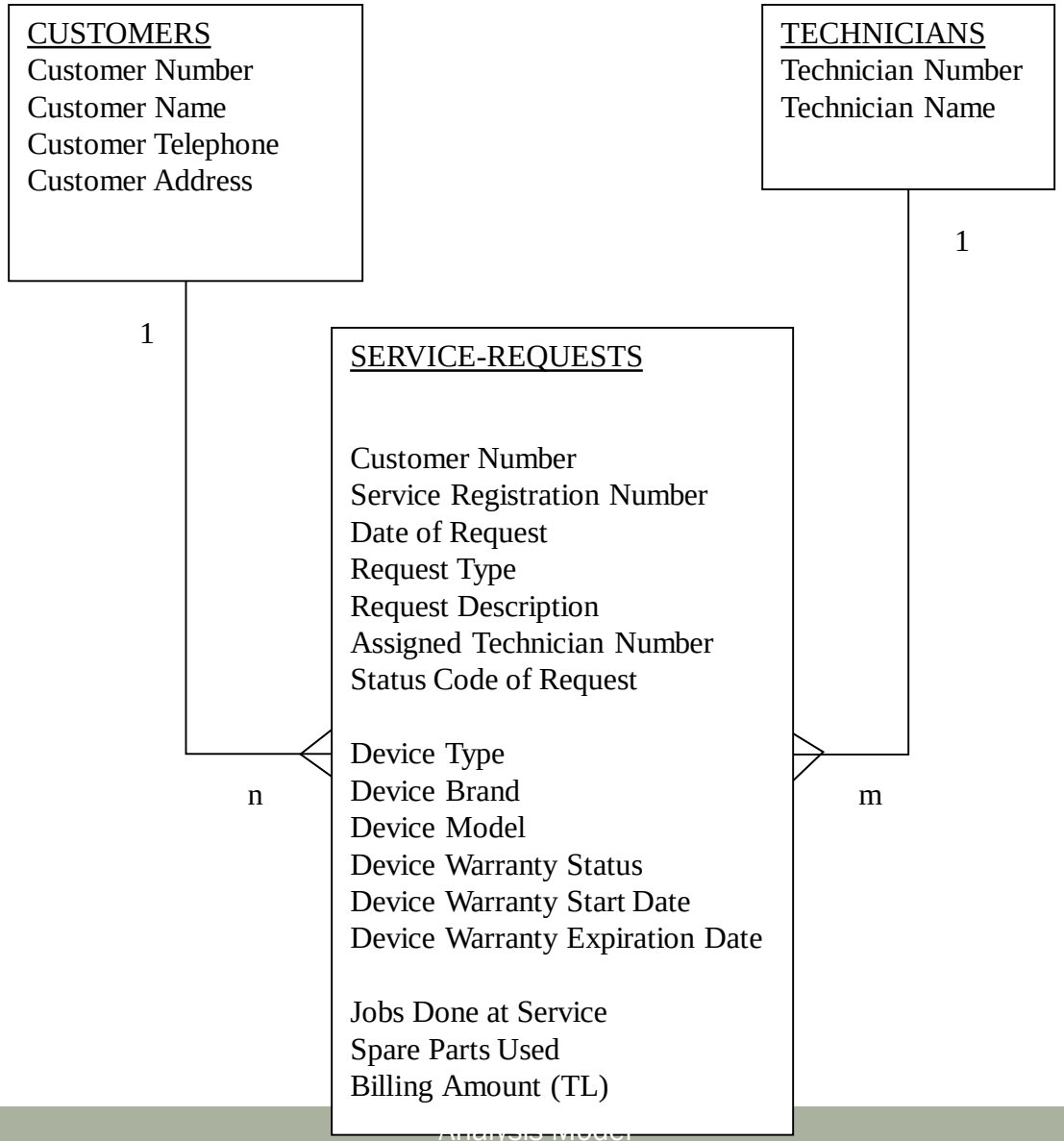
- For each service request the followings should be recorded: *Device information (device type, brand, model, warranty status, start date, expiration date); Jobs done at service, Spare parts used if any, Billing amount (TL).*

- For customers who has warranty agreement, periodic maintainances will be tracked. For this purpose, a list of devices which are sorted by warranty expiration date should be available.

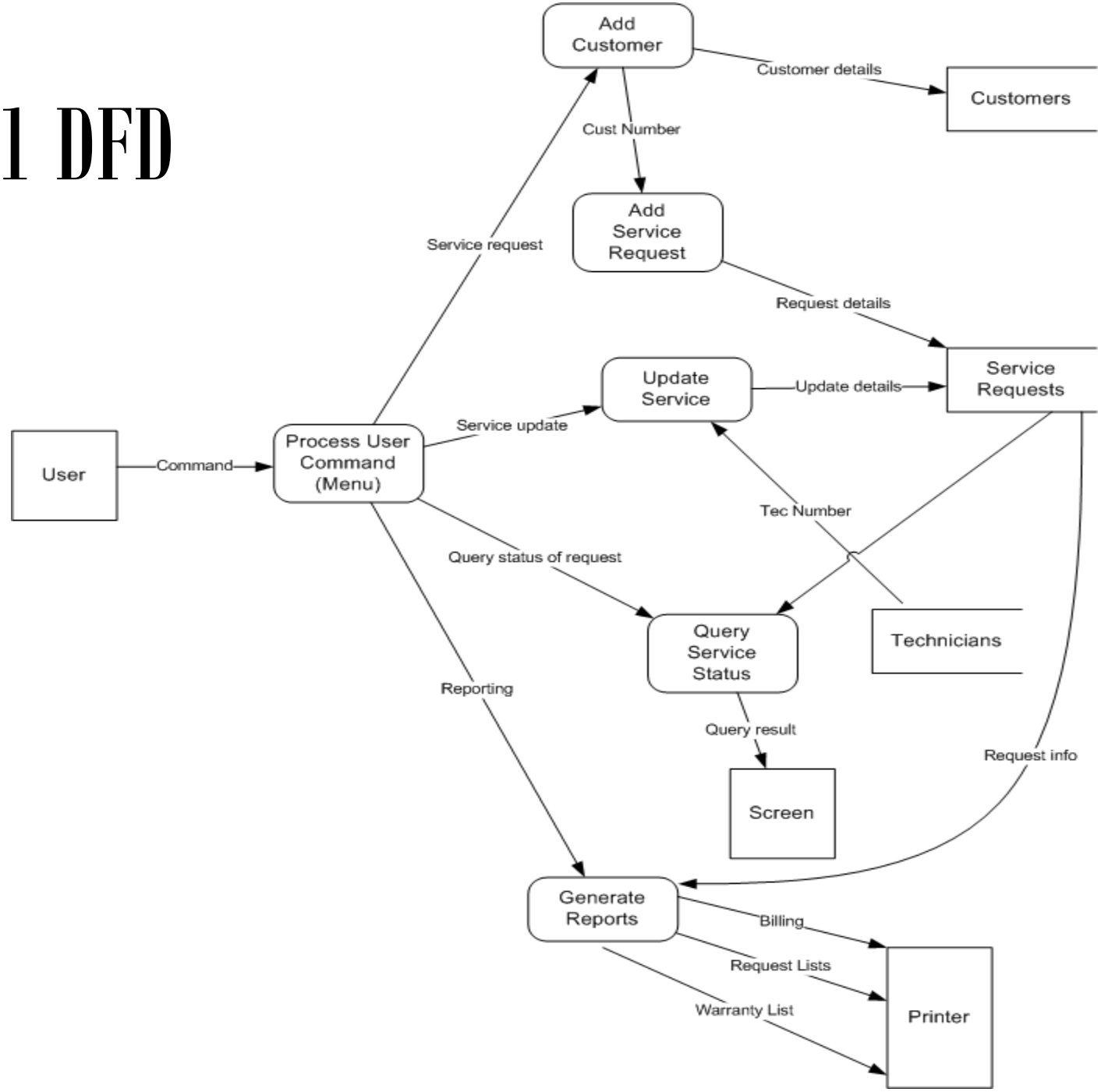
Statement of Software Scope (4)

- "Service Request Lists" should be available with different criteria:
  - by service registration number
  - by customer name
  - by status code
  - by device type
  - by request type
  - by date of request
  - by technician name

Entity Relationship Diagram (ERD)



Level-1 DFD


Add Customer
Customer details
Customers
Cust Number
Add Service Request
Service request
Request details
Update Service
Update details
Service Requests
User
Command
Process User Command (Menu)
Service update
Tec Number
Query status of request
Query Service Status
Technicians
Reporting
Query result
Request info
Screen
Generate Reports
Billing
Request Lists
Warranty List
Printer

Supplementary Material

- From the Book ‘Object-Oriented Software Engineering: Using UML, Patterns, and Java’ by Bruegge and Detoit

Wrap-up

This week we present

- System Engineering: How to model and understand the overall components of a software system
- Requirements Engineering: How to manage and acquire the needs of customer
- Requirement Process: What phases should be applied in effectively gathering requirements.
- Structural Analysis: Where the main focus of the analysis stage is handling static and dynamic system behaviour separately