Software Engineering

Dr. Mohammed Badawy

mbmbadawy@yahoo.com
mohamed.badawi@el-eng.menofia.edu.eg
Room: 417,   Ext. 7332

13 May 2016

Chapter 3

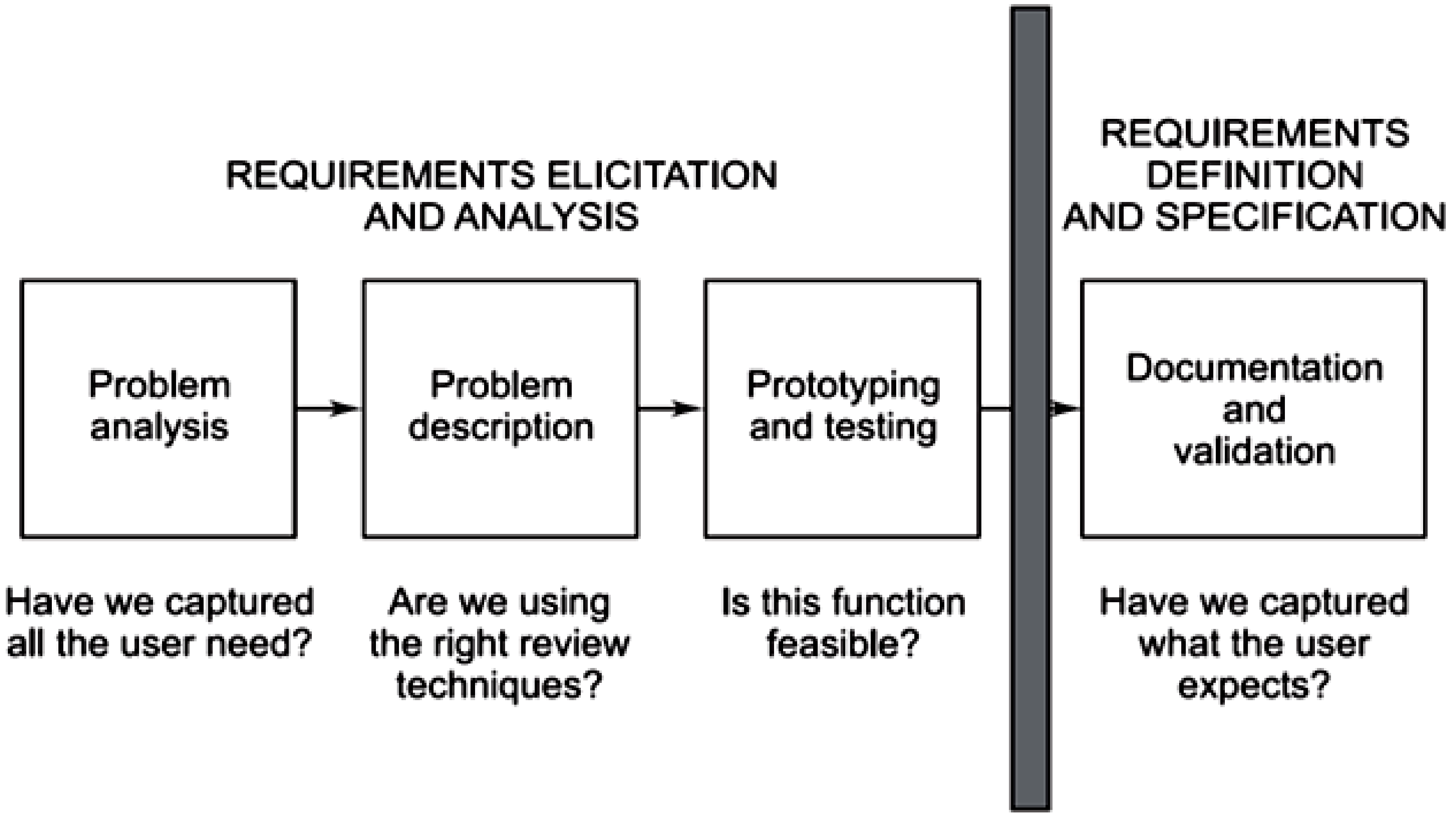
Software Requirements Analysis and Specification

Contents

Introduction

- A requirement is a feature of the system or a description of something the system is capable of doing in order to fulfil the system’s purpose.
- Requirements describe the “what” of a system, not the “how.”
- Requirements engineering produces one large document, written in a natural language.
- The next Figure illustrates the process of determining the requirements for a software-based system.

Introduction



REQUIREMENTS ELICITATION AND ANALYSIS
REQUIREMENTS DEFINITION AND SPECIFICATION
Problem analysis
Problem description
Prototyping and testing
Documentation and validation
Have we captured all the user need?
Are we using the right review techniques?
Is this function feasible?
Have we captured what the user expects?

Requirement Engineering

- Is <u>the systematic use of proven principles, techniques, and language tools for </u>the cost-effective analysis, documentation, and on-going evaluation of the user's needs and the specifications of the external behaviour of a system to satisfy those user needs.
- The **input** to requirements engineering is the problem statement prepared by the customer.
- The **output** of the RE process is a system requirements specification called the Requirement Definition and Description (RDD).

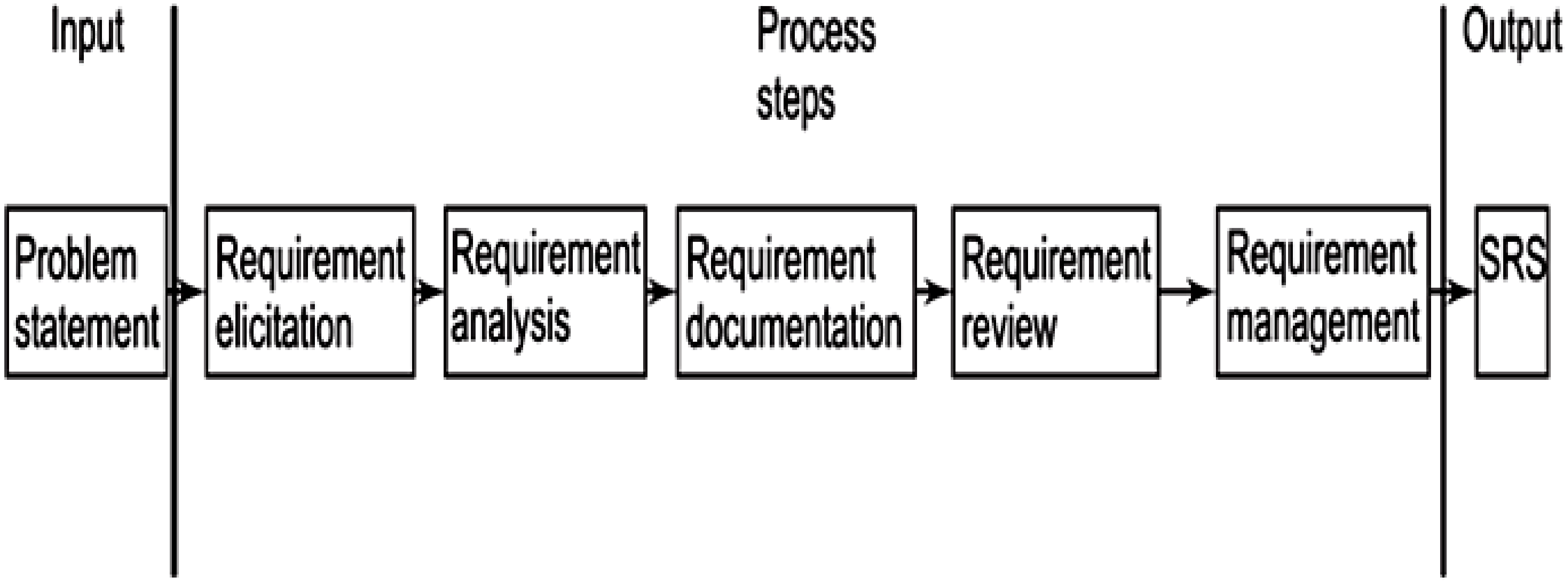
Types of Requirements

- On the basis of their functionality, the requirements are classified into the following two types:

1. **Functional requirements**: They define factors, such as I/O formats, storage structure, computational capabilities, timing, and synchronization
2. **Non-functional requirements**: They define the properties or qualities of a product including usability, efficiency, performance, reliability, portability, etc.

Process of Requirements Engineering

- The next Figure illustrates the process steps of requirements engineering.



Requirement Elicitation and Analysis

- The requirements are gathered from various sources. The sources are: Customer (Initiator), End Users, Primary Users, Secondary Users, and Stakeholders
- There are several practical approaches to requirements elicitation:
  - Joint application design (JAD)
  - Quality function deployment (QFD)
  - Designer as apprentice

What is JAD?

- JAD involves highly structured group meetings or mini-retreats with system users, system owners, and analysts in a single room for an extended period
- These meetings occur four to eight hours per day and over a period lasting one day to a couple of weeks
- Specifically, software engineers can use JAD for: Eliciting requirements and for the SRS, Design and software design description, Code, Tests and test plans, and User manuals

How do you plan for a JAD session?

- Planning for a review or audit session involves three steps:
  - Selecting participants
  - Preparing the agenda
  - Selecting a location
- Reviews and audits may include some or all of the following participants: Sponsors (for example, senior management) - A team leader - Users and managers who have ownership of requirements and business rules – Scribes - Engineering staff

How do you plan for a JAD session?

- The sponsor, analysts, and managers select a leader
- The leader may be in-house or a consultant
- One or more scribes (note-takers) are selected, normally from the software development team
- Before planning a session, the analysts and sponsor determine the scope of the project and set the high-level requirements of each session
- The agenda, code, and documentation is then sent to all participants well in advance of the meeting so that they have sufficient time to review them, make comments, and prepare questions

Ground rules for JAD sessions

- Stick to the agenda
- Stay on schedule
- Ensure that the scribe is able to take notes
- Avoid technical jargon
- Resolve conflicts (try not to defer them)
- Encourage group consensus
- Encourage user and management participation without allowing individuals to dominate the session
- Keep the meeting impersonal

Requirement Analysis

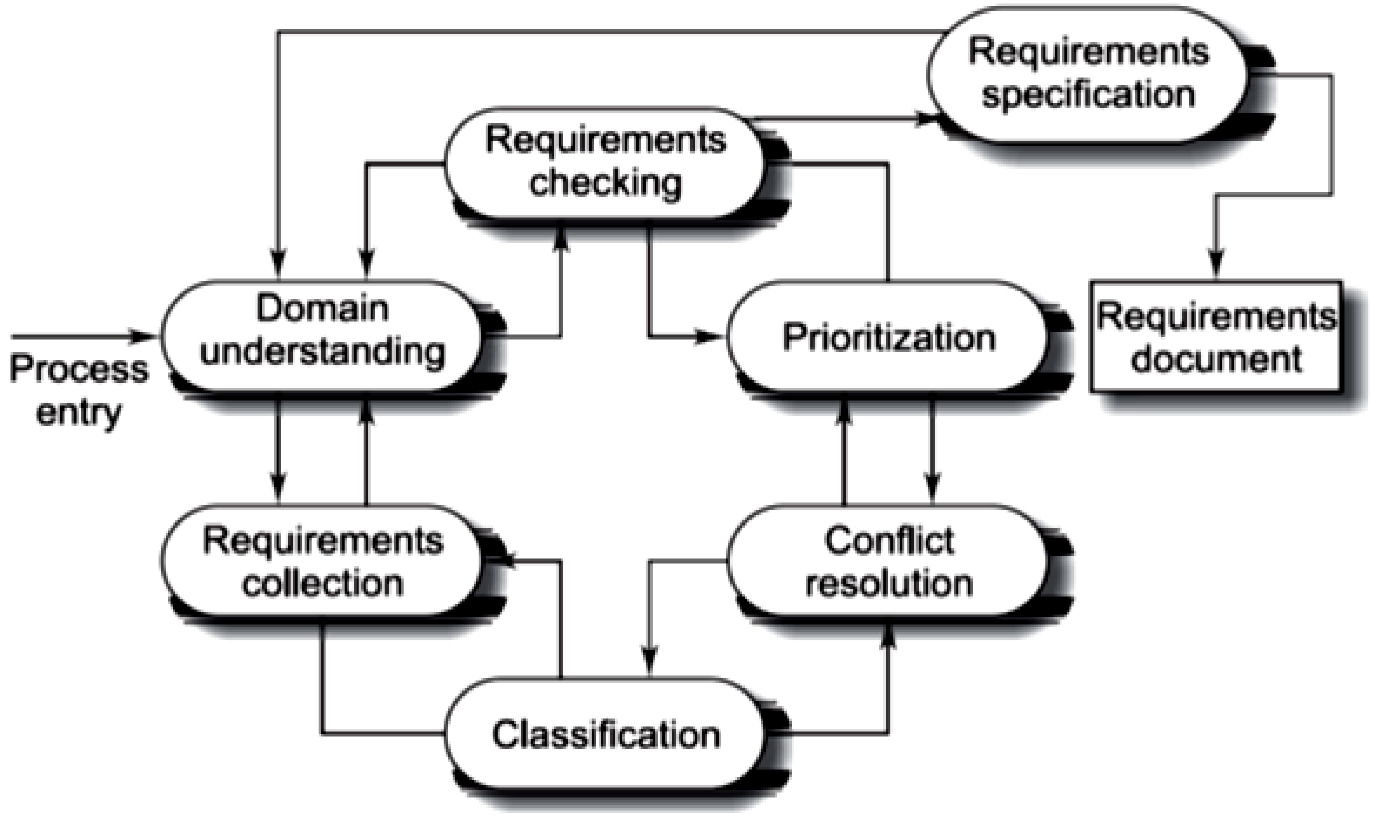
- Requirement analysis is a very important and essential activity after elicitation.
- In this phase, each requirement is analyzed from the point-of-view of validity, consistency, and feasibility for firm consideration in the RDD and then in the SRS.
- Validity confirms its relevance to goals and objectives and consistently confirms that it does not conflict with other requirements but supports others where necessary.
- Feasibility ensures that the necessary inputs are available without bias and error, and technology support is possible to execute the requirement as a solution.

Requirement Analysis

- The second portion of analysis attempts to find for each requirement, its functionality, features, and facilities and the need for these under different conditions and constraints.
- Functionality states “how to achieve the requirement goal.” Features describe the attributes of functionality, and facilities provide its delivery, administration, and communication to other systems.

Process Model of Elicitation and Analysis

- A generic process model of the elicitation and analysis process is shown in the next Figure



Process Model of Elicitation and Analysis

1. **Domain Understanding**: Analysts must develop their understanding of the application domain. For example, if a system for a supermarket is required, the analyst must find out how supermarkets operate.
2. **Requirements Collection**: This is the process of interacting with stakeholders in the system to discover their requirements. Obviously, domain understanding develops further during this activity.
3. **Classification:** This activity takes the unstructured collection of requirements and organizes them into coherent clusters.`

Process Model of Elicitation and Analysis

4. **Conflict Resolution**: Where multiple stakeholders are involved, requirements will conflict. This activity is concerned with findings and resolving these conflicts.
5. **Prioritization:** In any set of requirements some priorities will be more important than others. This stage involves interaction with stakeholders to discover the most important requirements.
6. **Requirements Checking**: The requirements are checked to discover if they are complete, consistent, and in accordance with what stakeholders really want from the system.

Requirements Documentation

- Requirements documentation is a very important activity, which is written after the requirements elicitation and analysis
- The requirements document is called the Software Requirements Specification (SRS)

Requirements Review

- Informal Review
- Formal Review
  - Verifiability
  - Comprehensibility
  - Traceability
  - Adaptability
- Contradictions, errors, and omissions in the requirements should be pointed out during the review and formally recorded
- It is then up to the users, the system procurer, and the system developer to negotiate a solution to these identified problems

Requirements Management

Is defined as a systematic approach to eliciting, organizing, and documenting the requirements of the system, and a process that establishes and maintains agreement between the customer and project team on the changing requirements of the system

Classes of Requirements

- **Enduring Requirements**: These are relatively stable requirements which derive from the core activity of the organization and which relate directly to the domain of the system
- **Volatile Requirements**: These are requirements which are likely to change during the system development or after the system has been put into operation

How can we rank requirements?

- We may use three levels of requirements:
  - Mandatory
  - Desirable
  - Optional requirements
- Mandatory requirements cannot be sacrificed
- Desirable requirements are important but could be sacrificed if necessary to meet schedule or budget
- Optional requirements would be nice to have, but are readily sacrificed

What does ranking requirements do for us?

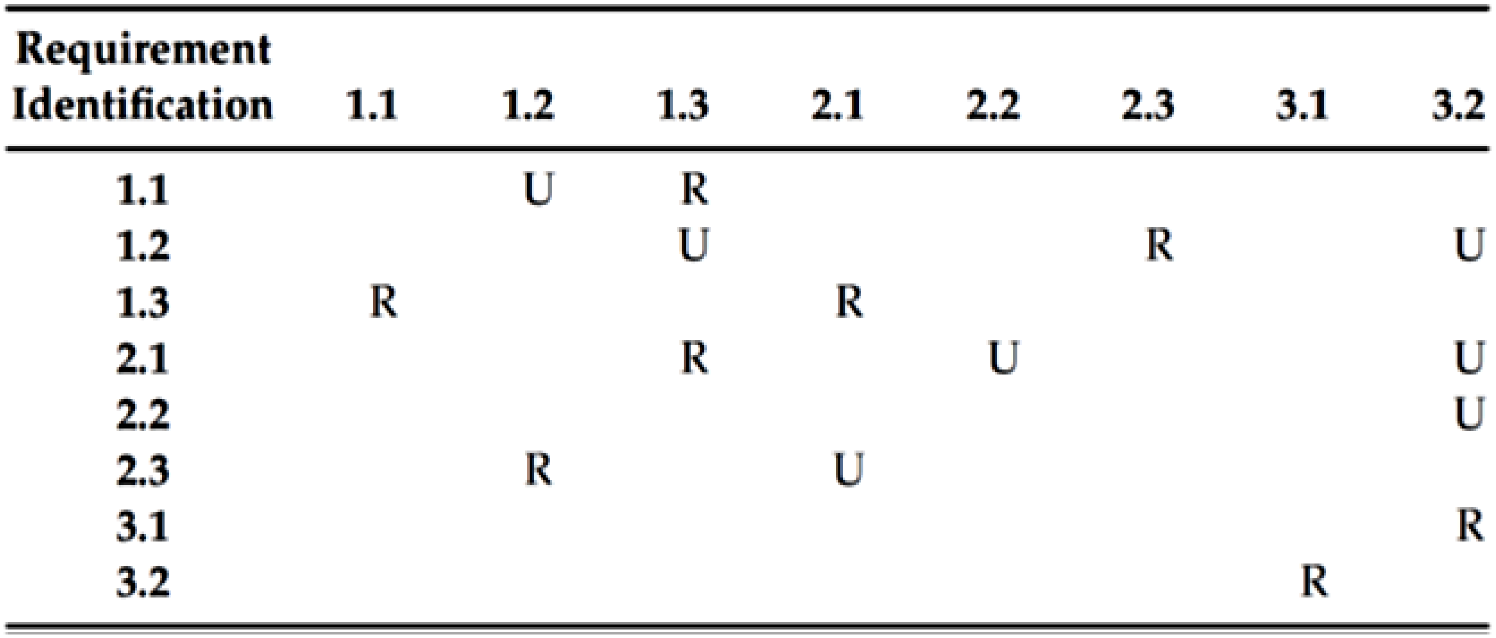
- Ranking requirements is quite helpful when tradeoffs need to be made
- For example, if time or work force is limited, then place the focus on the higher ranked requirements
- The same principle holds for testing; if testing time is limited, then it can be focused on the requirements pertaining to the higher-level requirements

What is requirements traceability?

- Is concerned with the relationships between requirements, their sources, and the system design
- Requirements can be linked to the source, to other requirements, and to design elements
- Source traceability links requirements to the stakeholders who proposed these requirements
- Requirements traceability links between dependent requirements
- Design traceability links from the requirements to the design

What does a traceability matrix look like?

- One type of traceability matrix is shown in Table below
- Requirements identification numbers label both the rows and columns.
- An “R” is placed in a corresponding cell if the requirement in that row references the requirement in that column.
- A “U” corresponds to an actual use dependency between the two requirements.



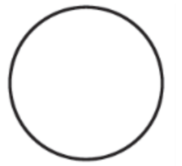
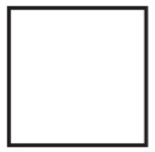
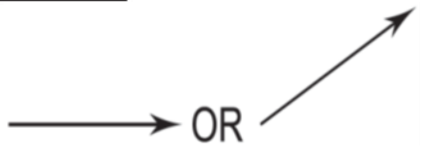
Requirement Identification	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		U	R					
1.2			U			R		U
1.3	R			R				
2.1			R		U			U
2.2								U
2.3		R		U				
3.1								R
3.2							R	

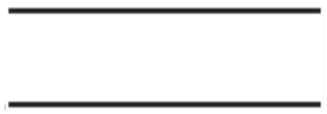
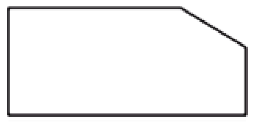
Data-Flow Diagrams DFDs

- Data-Flow Diagrams (DFD) are also known as data-flow graphs or bubble charts
- DFDs show the flow of data through a system
- It is an important modeling tool that allows us to picture a system as a network of functional processes
- Data-flow diagrams describe systems as collections of data that are manipulated by functions

Data-Flow Diagrams DFDs

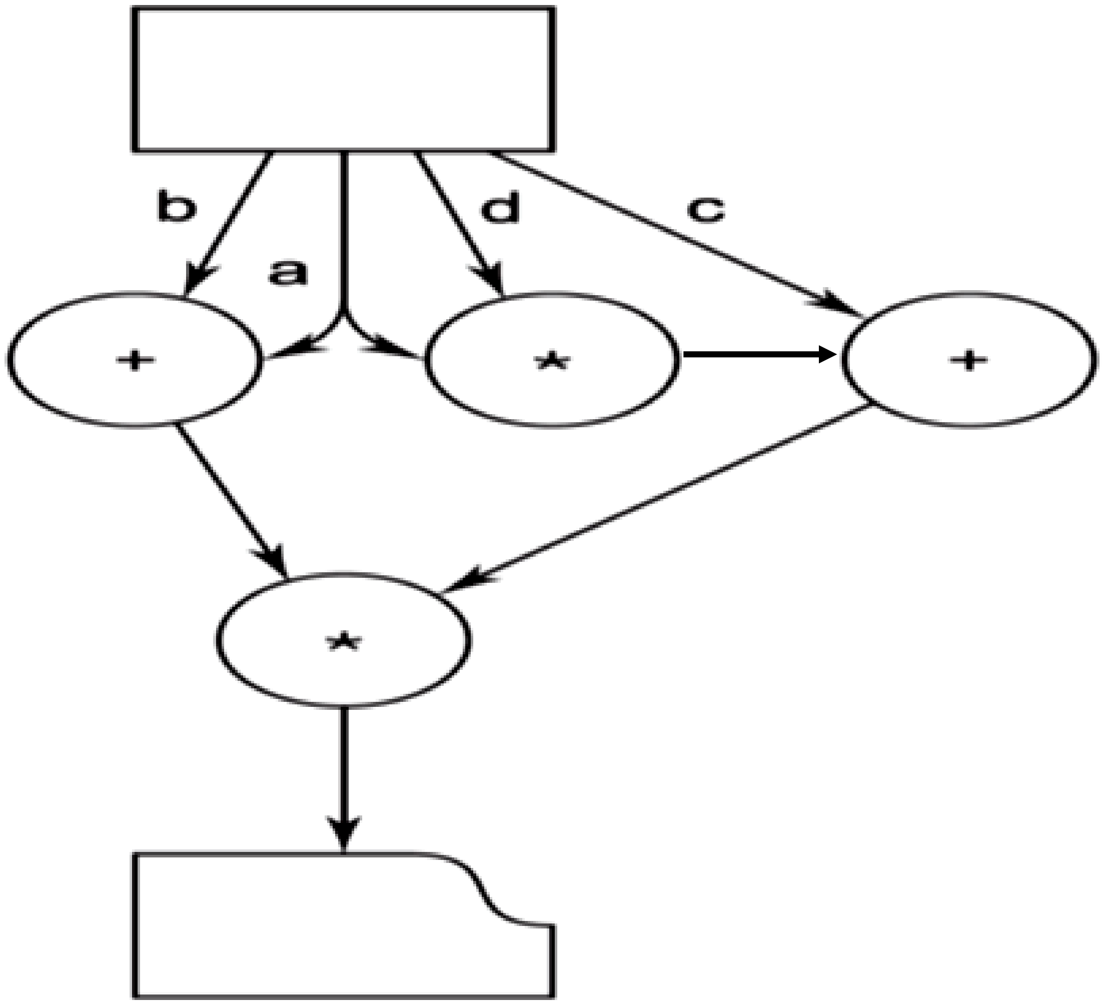
There are different types of symbols used to construct DFDs:

- Function symbol 
- External entity 
- Data-flow symbol 

- Data-store symbol 
- Output Symbol 

Example 1

- Construct a DFD that describes the arithmetic expression (a + b) * (c + a * d).
- Assume that the data a, b, c, and d are read from a keyboard and the result is printed

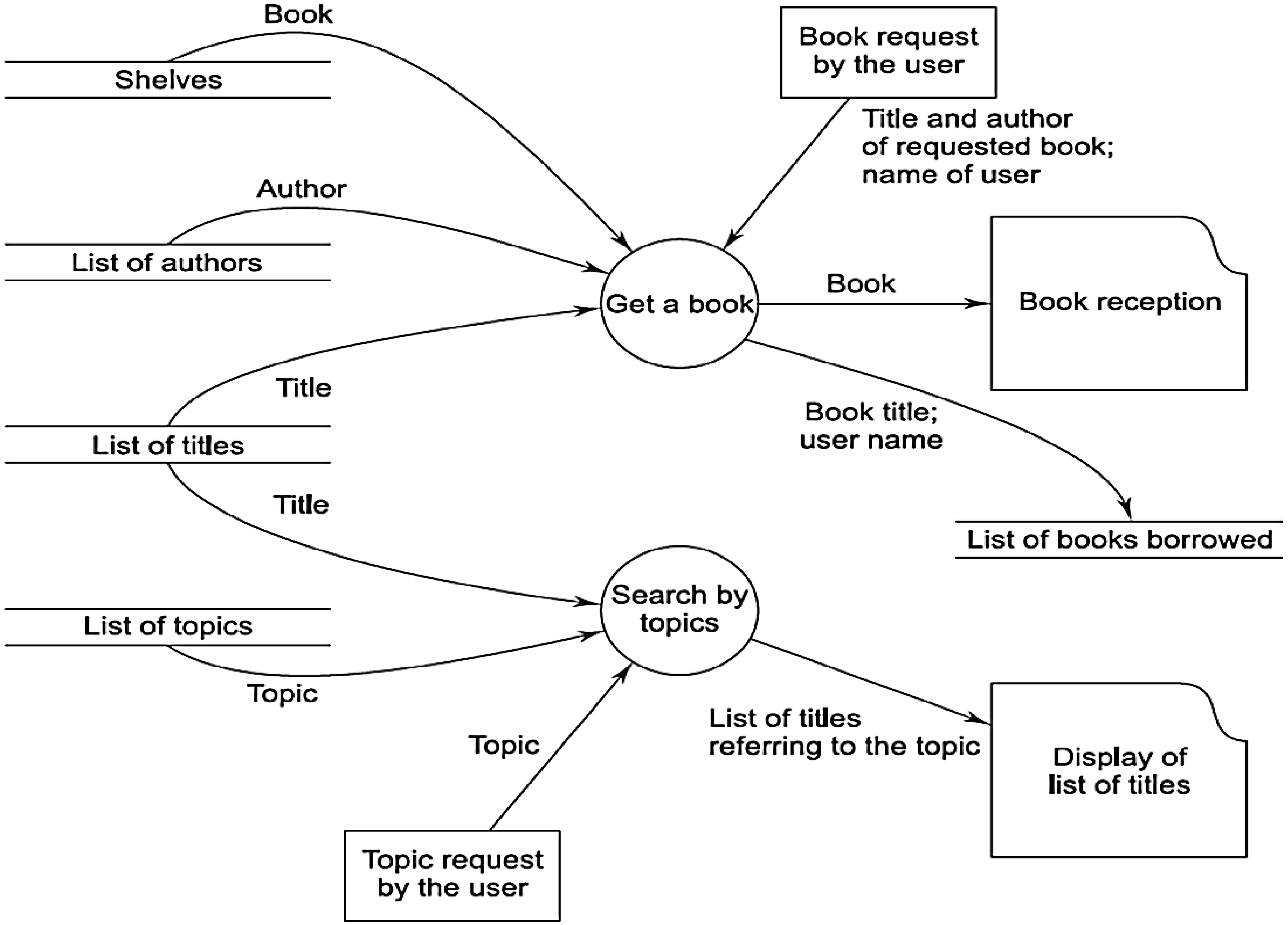


Example 2

- Construct a DFD that describes a simplified information system for a public library The data and functions shown are not necessarily computer data and computer functions
- The DFD describes physical objects, such as books and shelves, together with data stores that are likely to be, but are not necessarily, realized as computer files.
- Getting a book from the shelf can be done either automatically—by a robot—or manually.
- In both cases, the action of getting a book is represented by a function depicted by a bubble.

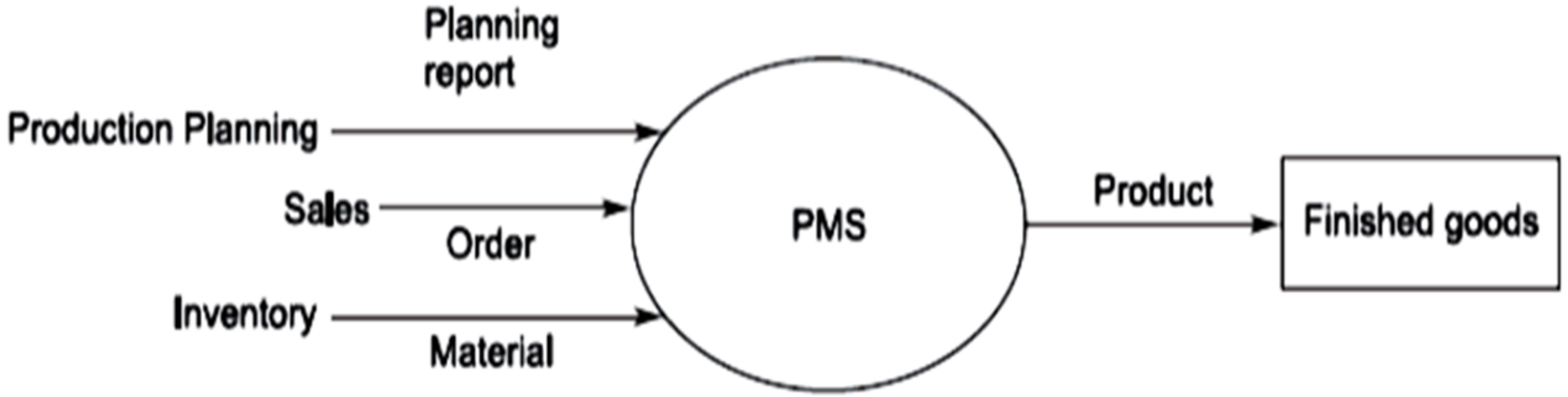
Example 2

- The figure could even represent the organization of a library with no computerized procedures.
- It describes the fact that, in order to obtain a book, the following are necessary:
  - an explicit user request consisting of the title and the name of the author of the book and the user's name;
  - access to the shelves that contain the books;
  - a list of authors; and a list of titles.
- These provide the information necessary to find the book.

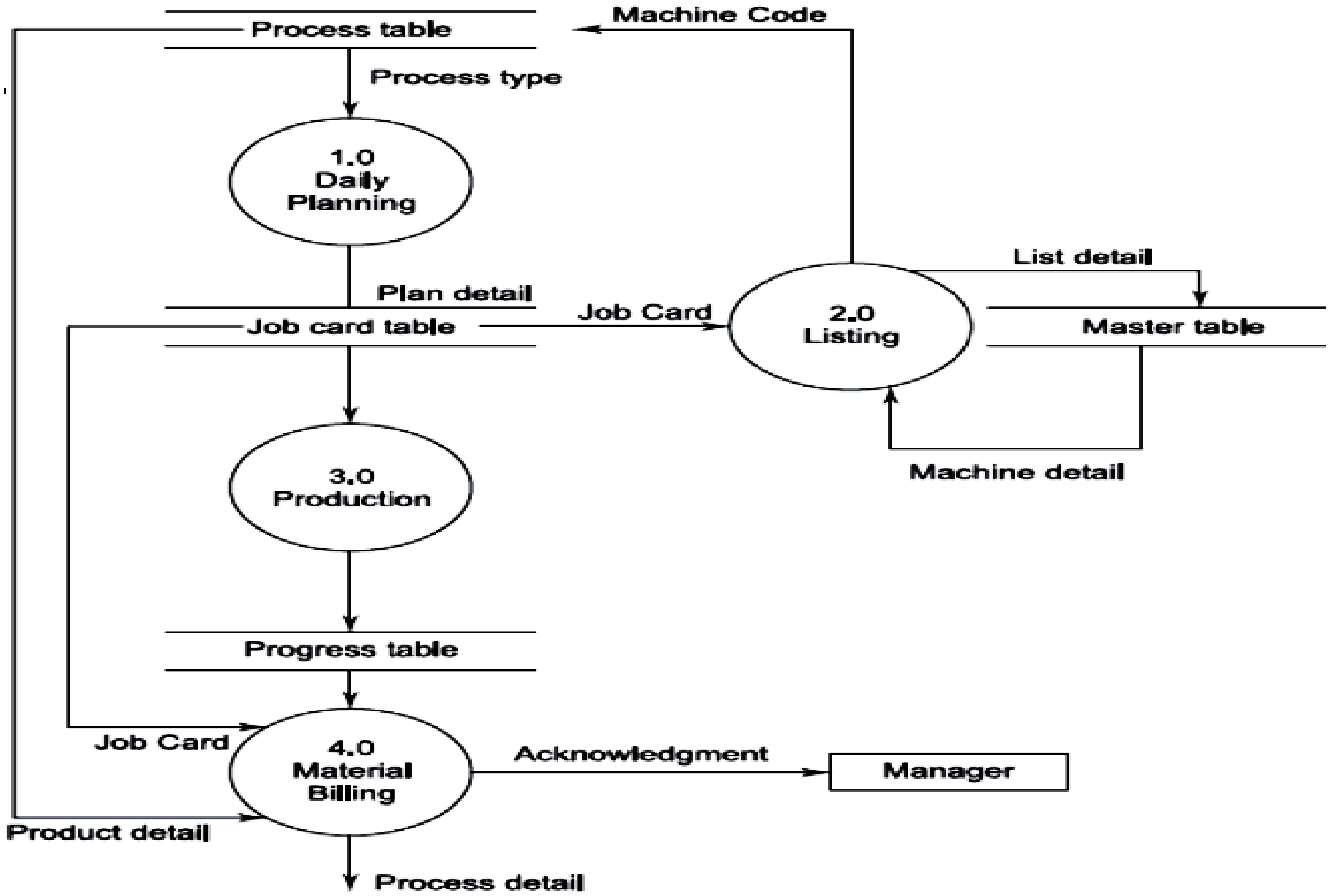

Book
Shelves
Book request by the user
Title and author of requested book; name of user
Author
List of authors
Get a book
Book
Book reception
Title
List of titles
Book title; user name
Title
List of books borrowed
Search by topics
List of topics
Topic
Topic
List of titles referring to the topic
Display of list of titles
Topic request by the user

Levels of A DFD

- The initial level is called the context level or fundamental system model or a 0-level DFD
- If we expand the 0-level processes then we get the 1st-level DFD and if we further expand the 1st-level processes then we get the 2nd-level DFD and so on
- The 0th and 1st levels of the DFD of a Production Management System are shown in the next figures



Example of A DFD



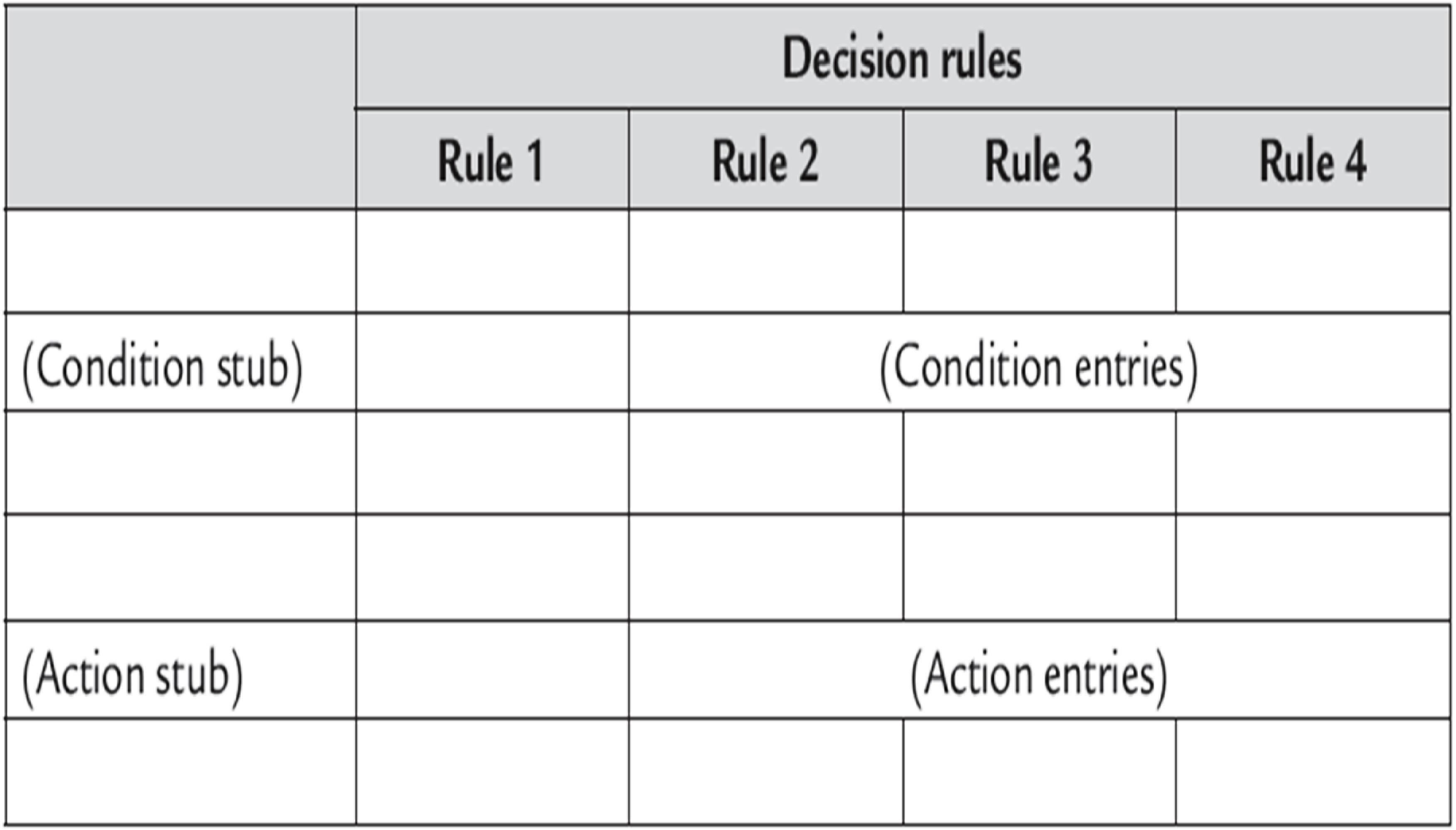
General Guidelines for Constructing DFDs

- ❖ Remember that a DFD is not a flowchart
- ❖ All names should be unique
- ❖ Processes are always running; they do not start or stop
- ❖ All data flows are named
- ❖ Give unique names to all the processes and external entities
- ❖ Avoid complex DFDs (if possible)
- ❖ Every process should have a minimum of one input and one output
- ❖ Only data needed to perform the process should be an input to the process
- ❖ The direction of flow is from source to destination

Decision Tables

- Decision tables provide a mechanism for recording complex decision logic
- Decision tables are widely used in data-processing applications and have extensively developed literature.
- A decision table is segmented into four quadrants:
  - Condition stub
  - Condition entry
  - Action stub
  - Action entry

Decision Tables

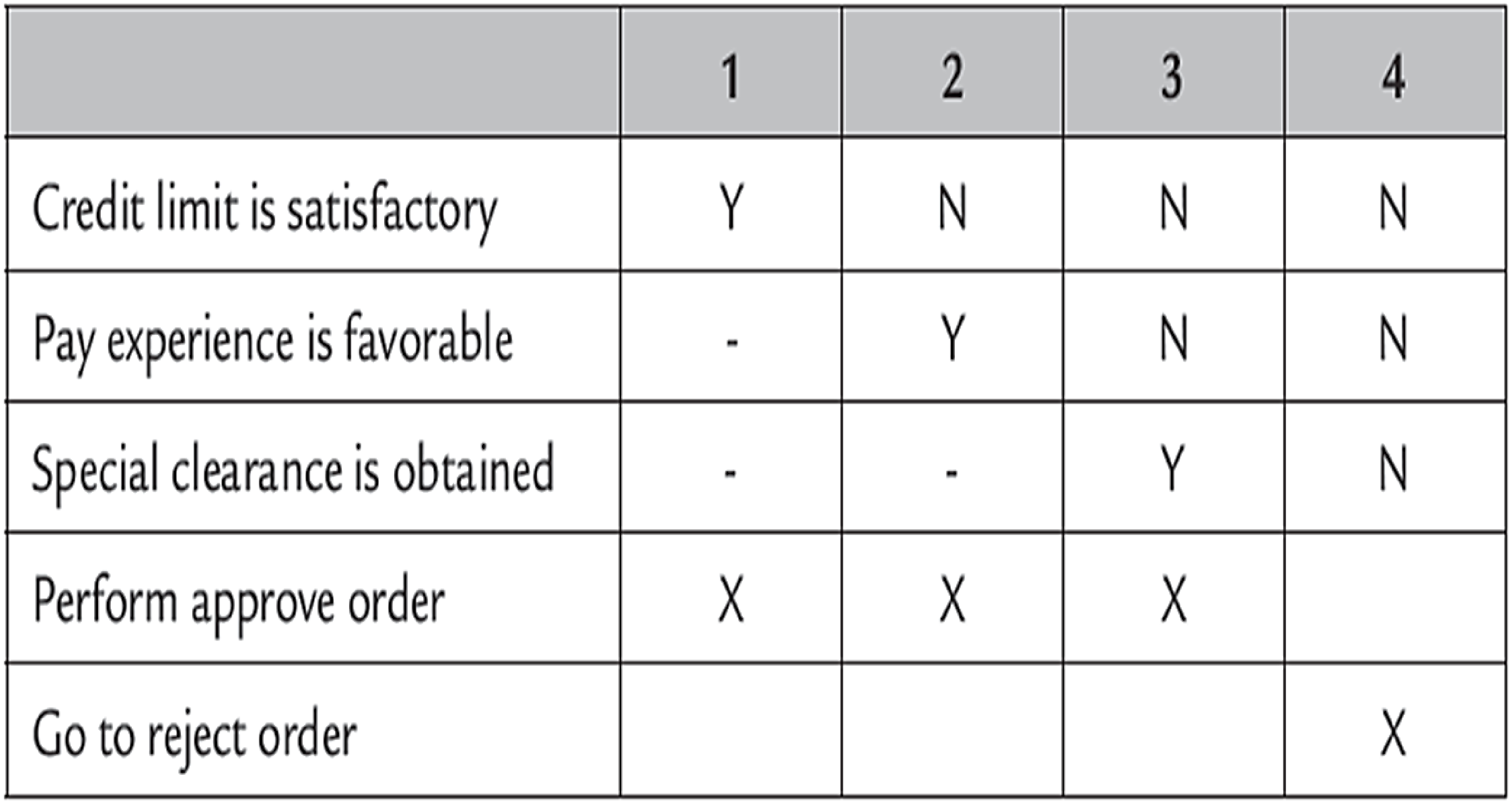


	Decision rules			
	Rule 1	Rule 2	Rule 3	Rule 4
(Condition stub)		(Condition entries)		
(Action stub)		(Action entries)		

Decision Tables

- The **condition stub** contains all of the conditions being examined.
- **Condition entries** are used to combine conditions into decision rules.
- The **action stub** describes the actions to be taken in response to decision rules
- **Action entry** quadrant relates decision rules to actions.
- The next table illustrates the format of a limited-entry decision table (entries are limited to Y, N, -, and X).
- In a limited-entry decision table, Y denotes "yes," N denotes "no," - denotes "don't care," and X denotes "perform action."

Decision Tables- Basic Elements



	1	2	3	4
Credit limit is satisfactory	Y	N	N	N
Pay experience is favorable	-	Y	N	N
Special clearance is obtained	-	-	Y	N
Perform approve order	X	X	X	
Go to reject order				X

Decision Tables- Basic Elements

- According to the previous table, orders are approved if the credit limit is not exceeded, or if the credit limit is exceeded but past experience is good, or if a special arrangement has been made.
- If none of these conditions hold, the order is rejected.
- The (Y, N, -) entries in each column of the condition entry quadrant form a decision rule.

An Ambiguous Decision Table

- If more than one decision rule has identical (Y, N, -) entries, the table is said to be ambiguous.
- Ambiguous pairs of decision rules that specify identical actions are said to be redundant, and those specifying different actions are contradictory.
- Contradictory rules permit specification of non-deterministic and concurrent actions.
- The next table illustrates redundant rules (R3 and R4) and contradictory rules (R2 and R3, and R2 and R4).

	Decision rules			
	Rule 1	Rule 2	Rule 3	Rule 4
C1	Y	Y	Y	Y
C2	Y	N	N	N
C3	N	N	N	N
A1	X			
A2		X		
A3			X	X

An Incomplete Decision Table

- A decision table is complete if every possible set of conditions has a corresponding action prescribed
- There are 2**N combinations of conditions in a table that has N conditions entries
- Failure to specify an action for any one of the combinations results in an incomplete decision table
- In the next table, the combination (N, N, N) for conditions C1, C2, and C3 has no action specified.
- Note also that the condition (Y, Y, N) specifies both actions A1 and A2.
- These multiple specified actions may be desired, or they may indicate a specification error.

An Incomplete Decision Table

C1	Y		N
C2		Y	N
C3			Y
A1		X	
A2	X		
A3			X

N
N
N
?
?
?

An Incomplete Decision Table

- The previous table illustrates the use of a Karnaugh map to check a decision table for completeness and multiple specified actions.
- The specification is incomplete if there are any blank entries in the Karnaugh map.
- The specification is multiply specified if there are any multiple entries in the Karnaugh map.

Example

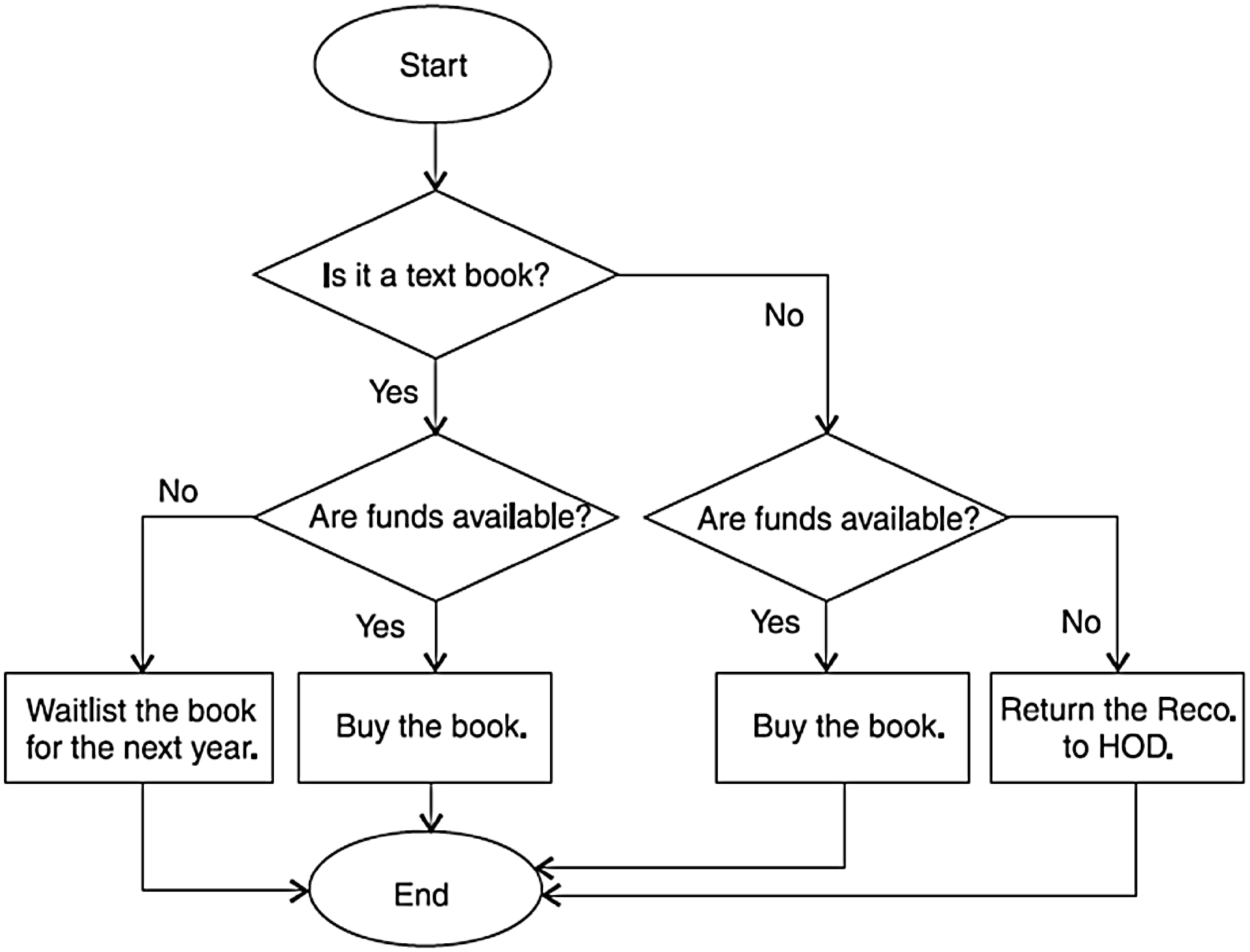
- A computer-based system performs two actions (A1 and A2) based on the status of 3 switches
- If at least one of the three switches is ON, it performs A1
- If at least two of the switches are ON, it performs A2
- Construct a decision table for the system
- Is the decision table complete? Why?
- Is there any contradictory or redundant rules? Why?

Example

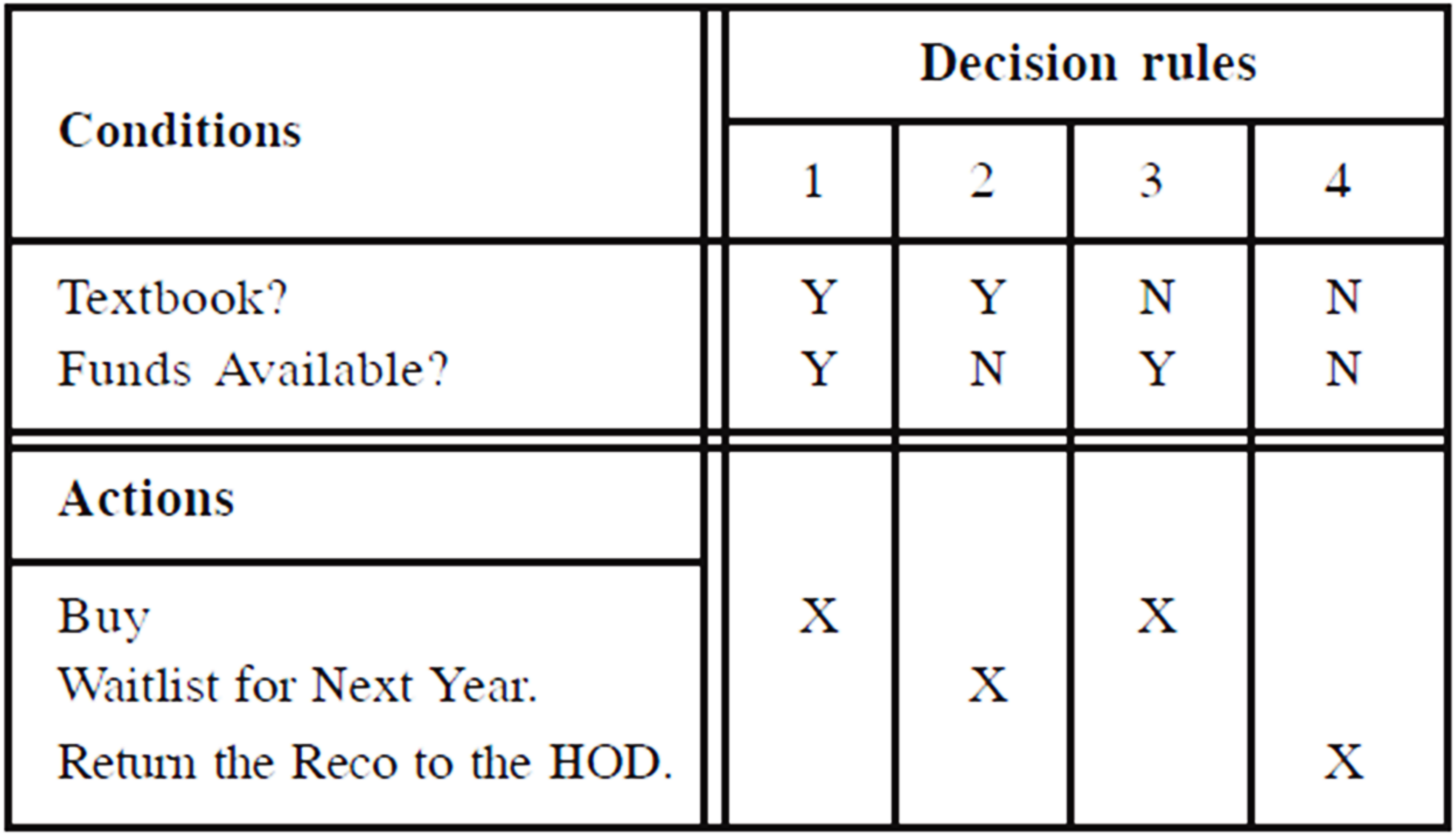
	1	2	3	4	5	6	7	8
SW1	N	Y	N	Y	N	Y	N	Y
SW2	N	N	Y	Y	N	N	Y	Y
SW3	N	N	N	N	Y	Y	Y	Y
A1		X	X	X	X	X	X	X
A2				X		X	X	X

Library Requisition

- The Head of the Department (HOD) recommends books to be bought by the Library
- If funds are available, then the books are bought
- In case funds don’t permit, a textbook is kept waitlisted for purchase during the next year, whereas the Library returns the requisitions for all other books to the Head of the Department


Start
Is it a text book?
Yes
No
No
Are funds available?
Yes
Are funds available?
Yes
No
Waitlist the book for the next year.
Buy the book.
Buy the book.
Return the Reco. to HOD.
End

Library Requisition Decision Table



Conditions	Decision rules			
	1	2	3	4
Textbook?	Y	Y	N	N
Funds Available?	Y	N	Y	N
Actions				
Buy	X		X	
Waitlist for Next Year.		X		
Return the Reco to the HOD.				X

Advantages of Decision Tables

- ❖ Decision rules are clearly structured
- ❖ Managers can be relieved from decision-making
- ❖ Consistency in decision-making
- ❖ Communication is easier between managers and analysts
- ❖ Documentation is easily prepared, changed, or updated
- ❖ Easy to use
- ❖ Easier to draw or modify compared to flowcharts
- ❖ Facilitate more compact documentation
- ❖ Easier to follow a particular path down one column than through complex and lengthy flowcharts

Disadvantages of Decision Tables

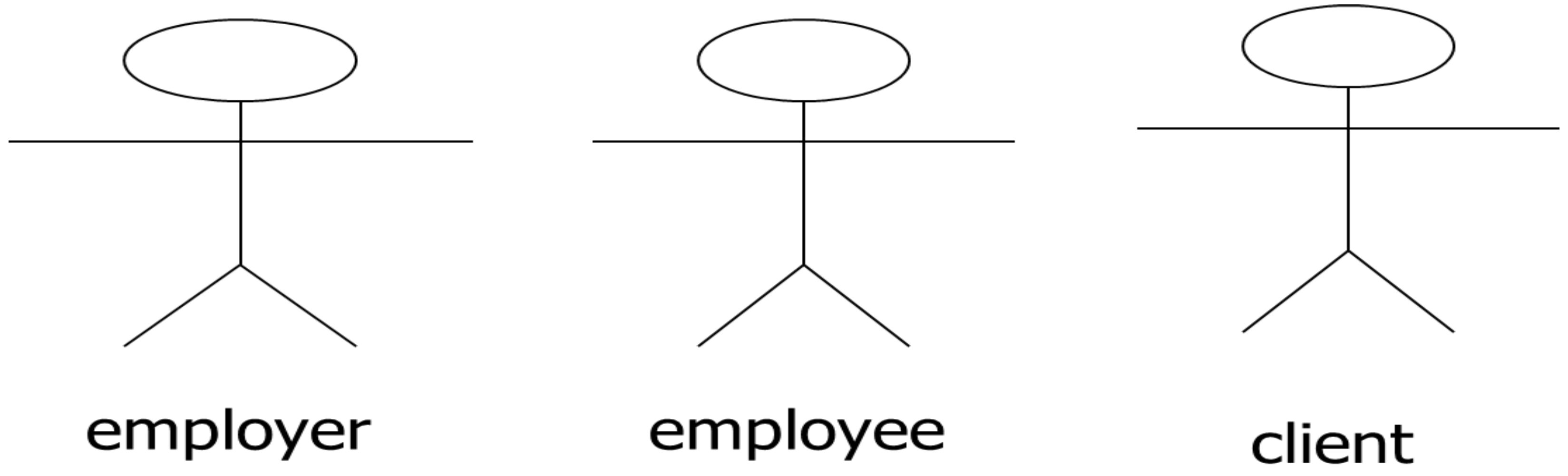
- Impose an additional burden
- Do not depict the flow
- Not easy to translate
- Cannot list all the alternatives

Use Case Diagram

- Getting started is the most difficulty part of any new process
- The first thing you need to do is understand what are you going to model and ultimately develop
- A use case diagram is an excellent way to communicate to management, customers, and other non-development people what a system will do when it is completed
- **A Use Case** <u>is a description of sequences of actions performed by a given system to produce a result for an actor</u>

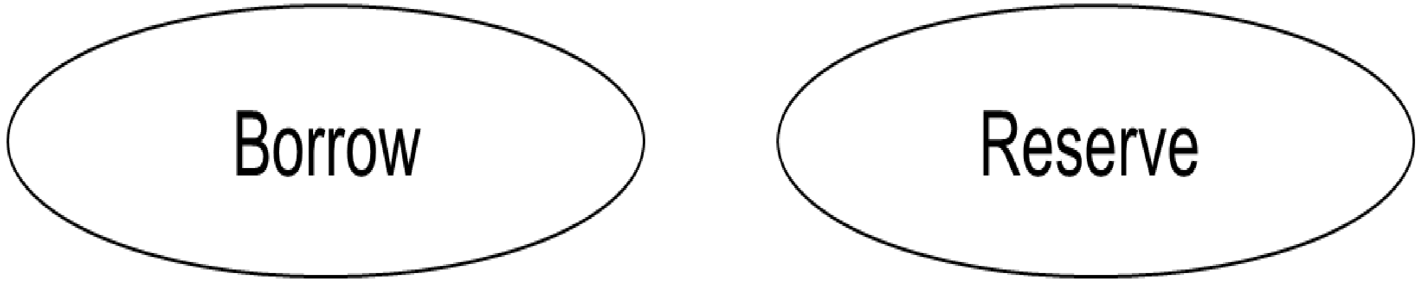
Actors

- Actors represent roles that humans, hardware devices, or external systems play while interacting with a given system
- They are not part of the system and are situated outside of the system boundary
- Actor is shown with a stick figure



Use Cases

- Use case is a particular activity a user can do on the system
- It is represented by an ellipse
- Two use cases for a library system shown below



University Record System (URS) Example

A University record system should keep information about its students and academic staff

- Students will have a list of subjects they are enrolled in.
- A student cannot be enrolled in any more than 10 subjects.
- Academic employees will have a salary, and a list of subjects they teach.
- An academic can teach no more than 3 subjects.

University Record System (URS) Example

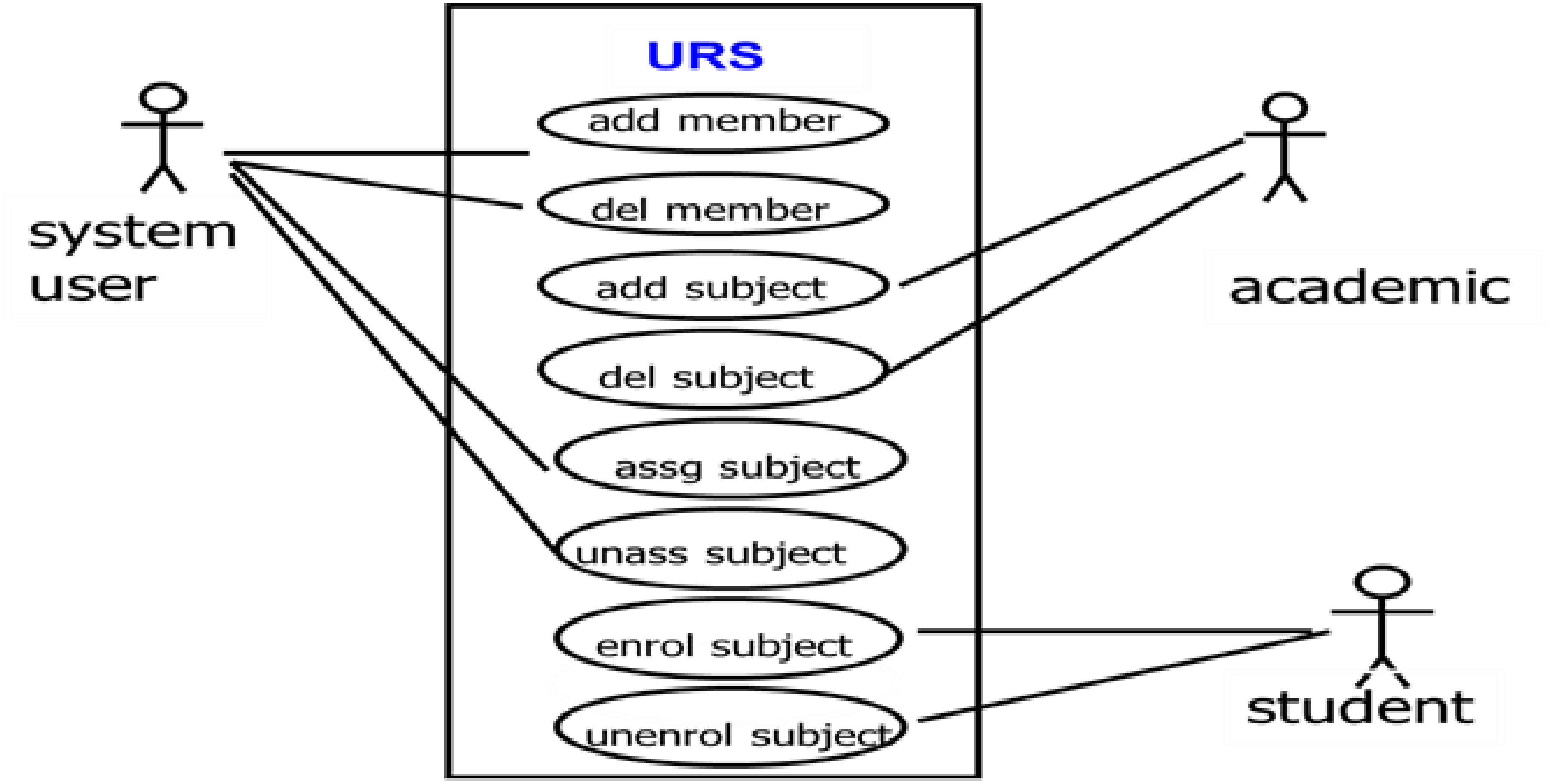
<u>Some Actions Supported by URS</u>

The system should be able to handle the following commands.

- Add and remove university members (students, and academic staff)
- Add and Delete subjects
- Assign and Un-assign subjects to students
- Assign and Un-assign subjects to academic staff.

University Record System (URS) Example

The use case diagram of the system



University Record System (URS) Example
Use Case Vs Scenarios

Each use case is one or more scenarios. Each scenario has a sequence of steps. For example:

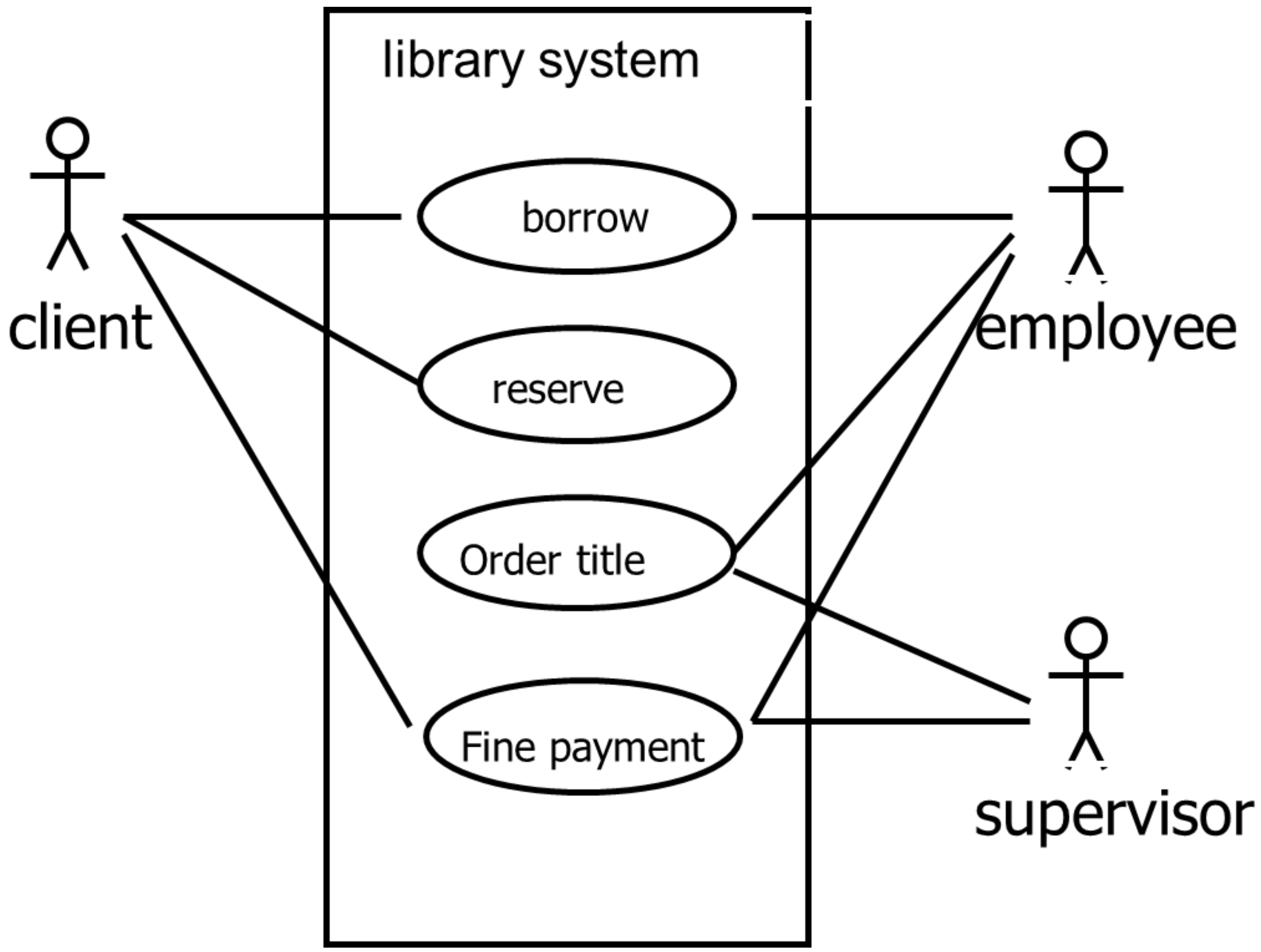
Add Subject Use Case:

- Scenario 1: Subject gets added successfully.
- Scenario 2: Adding the subject fails since the subject is already in the database.

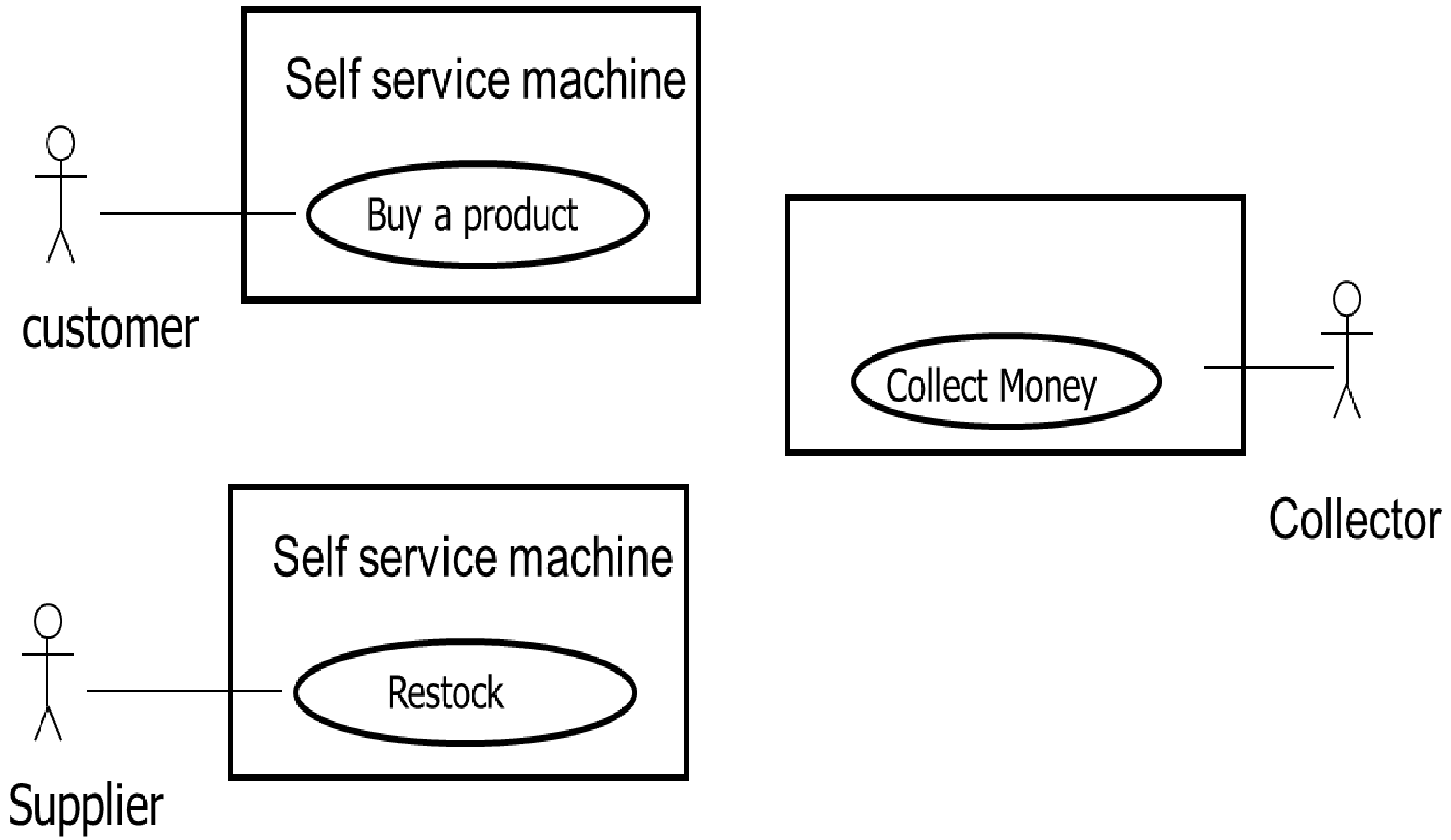
Enroll Subject Use Case:

- Scenario 1: Student is enrolled for the subject.
- Scenario 2: Enrollment fails since the student is already enrolled in the subject.

A library example



Use Case Diagrams Relationships



Use Case Diagrams Relationships

Inclusion

Inclusion enables to reuse one use case's steps inside another use case.

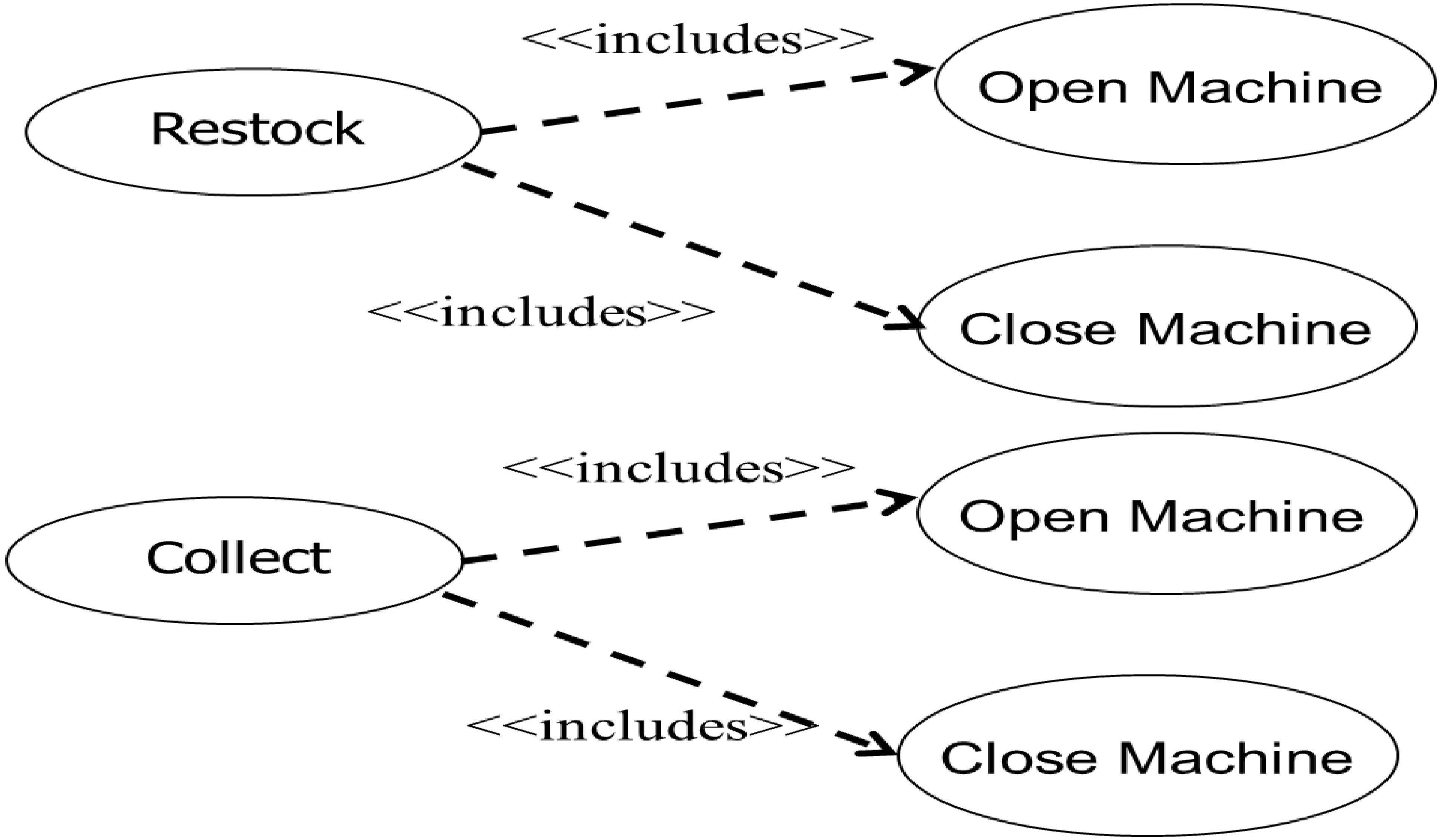
Extension

Allows creating a new use case by adding steps to existing use cases

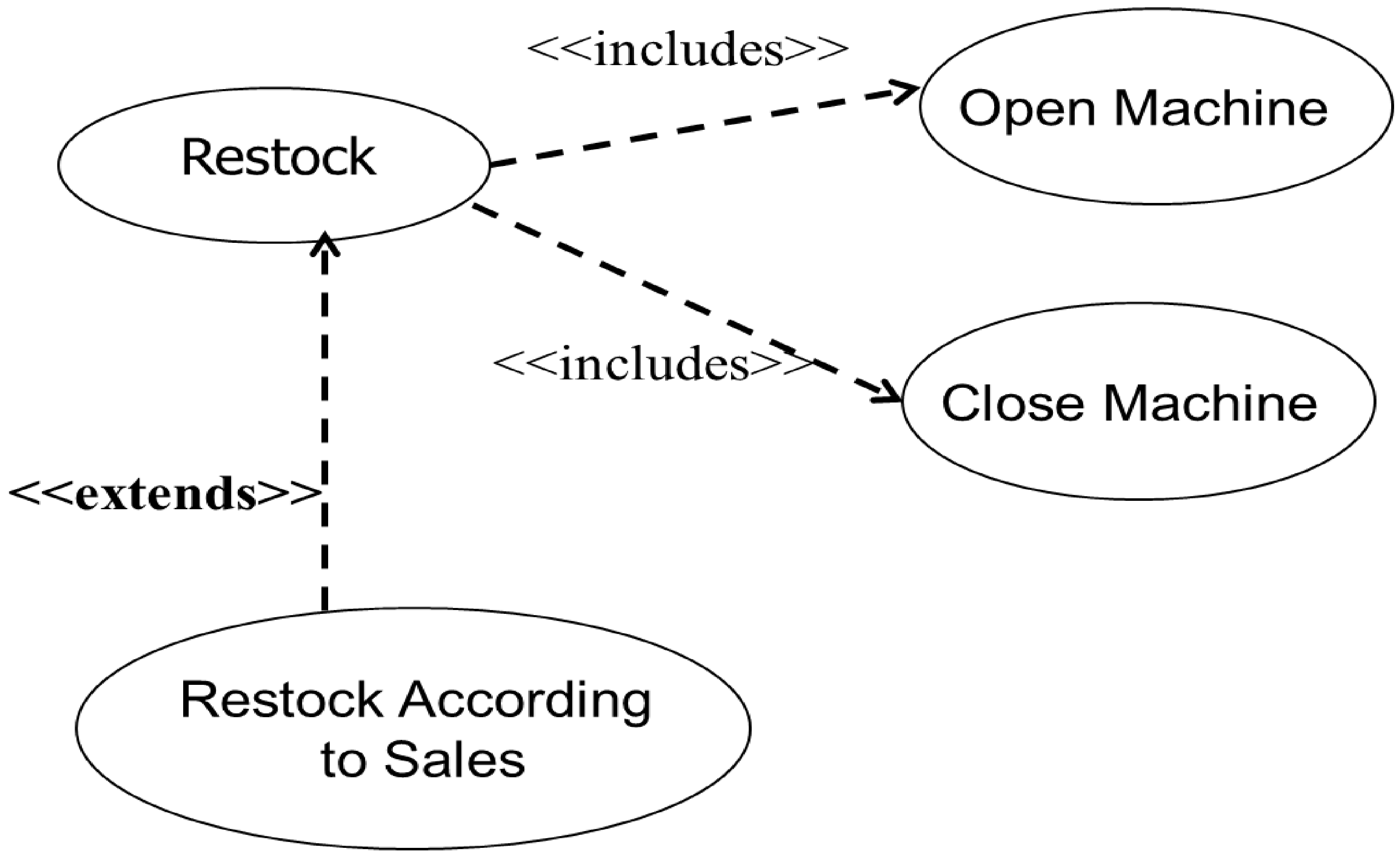
Generalization

Allows child use cases to inherit behavior from parent use cases

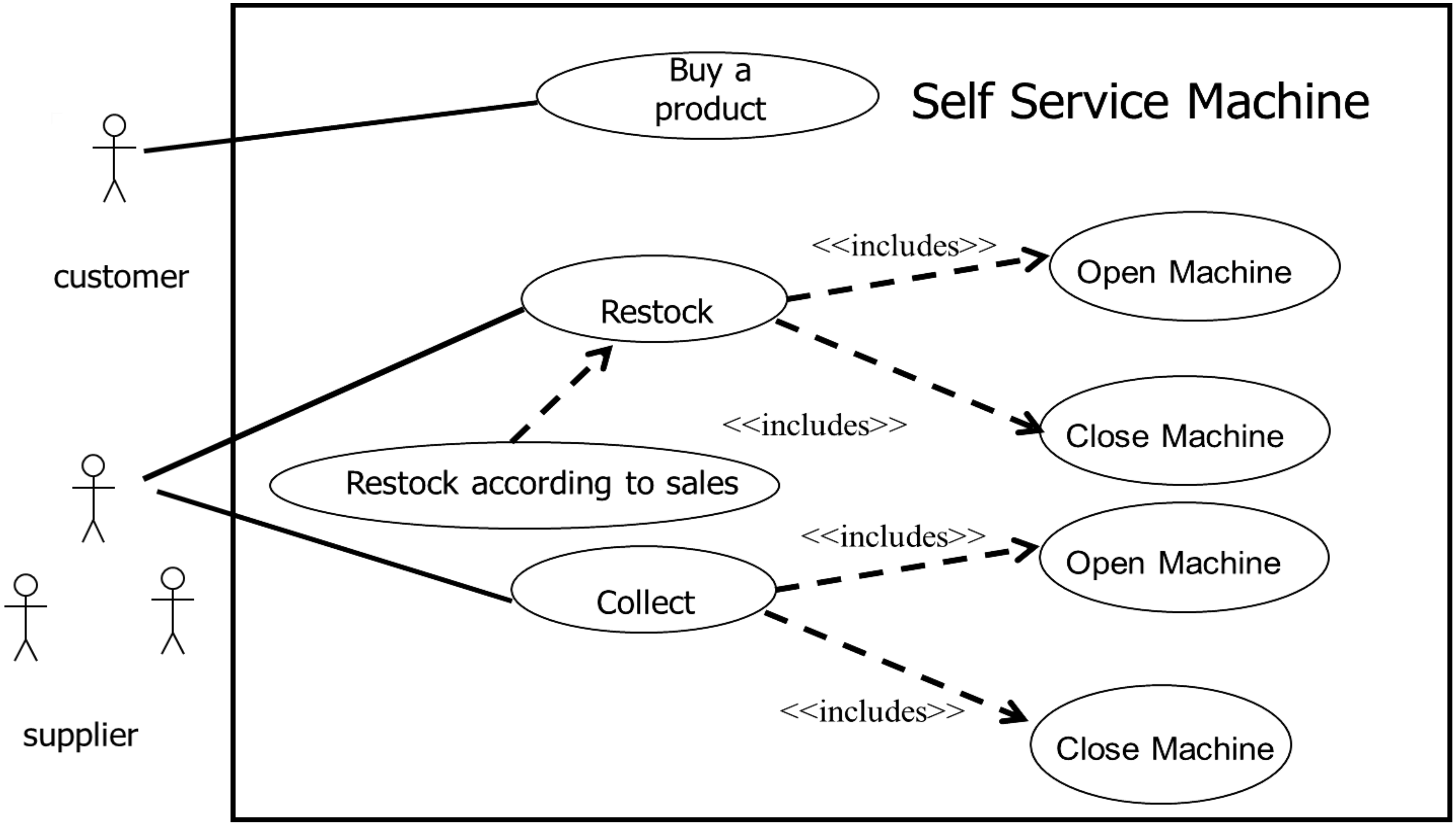
Use Case Diagrams Relationships



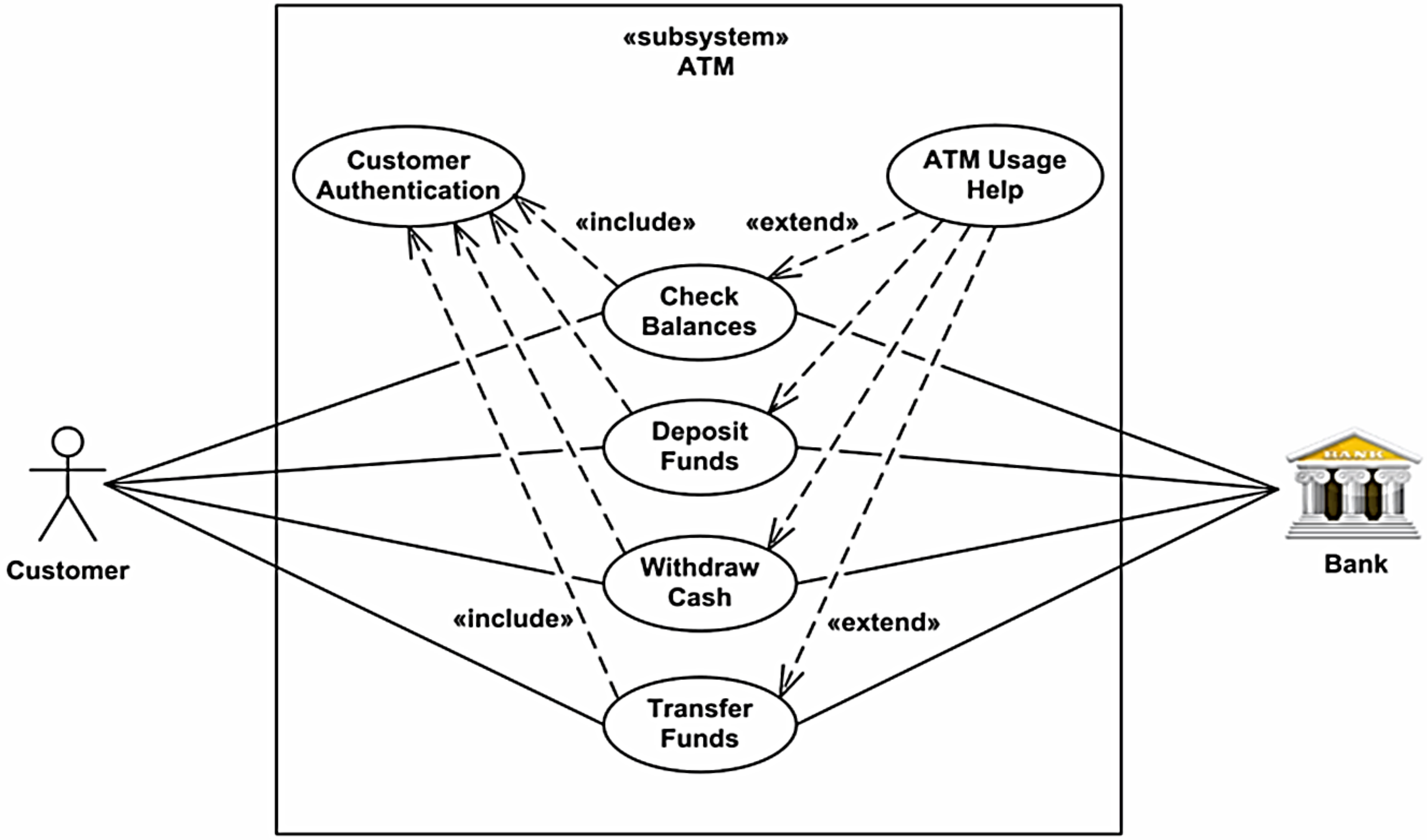
Use Case Diagrams Relationships



Self service machine Use Case Diagram

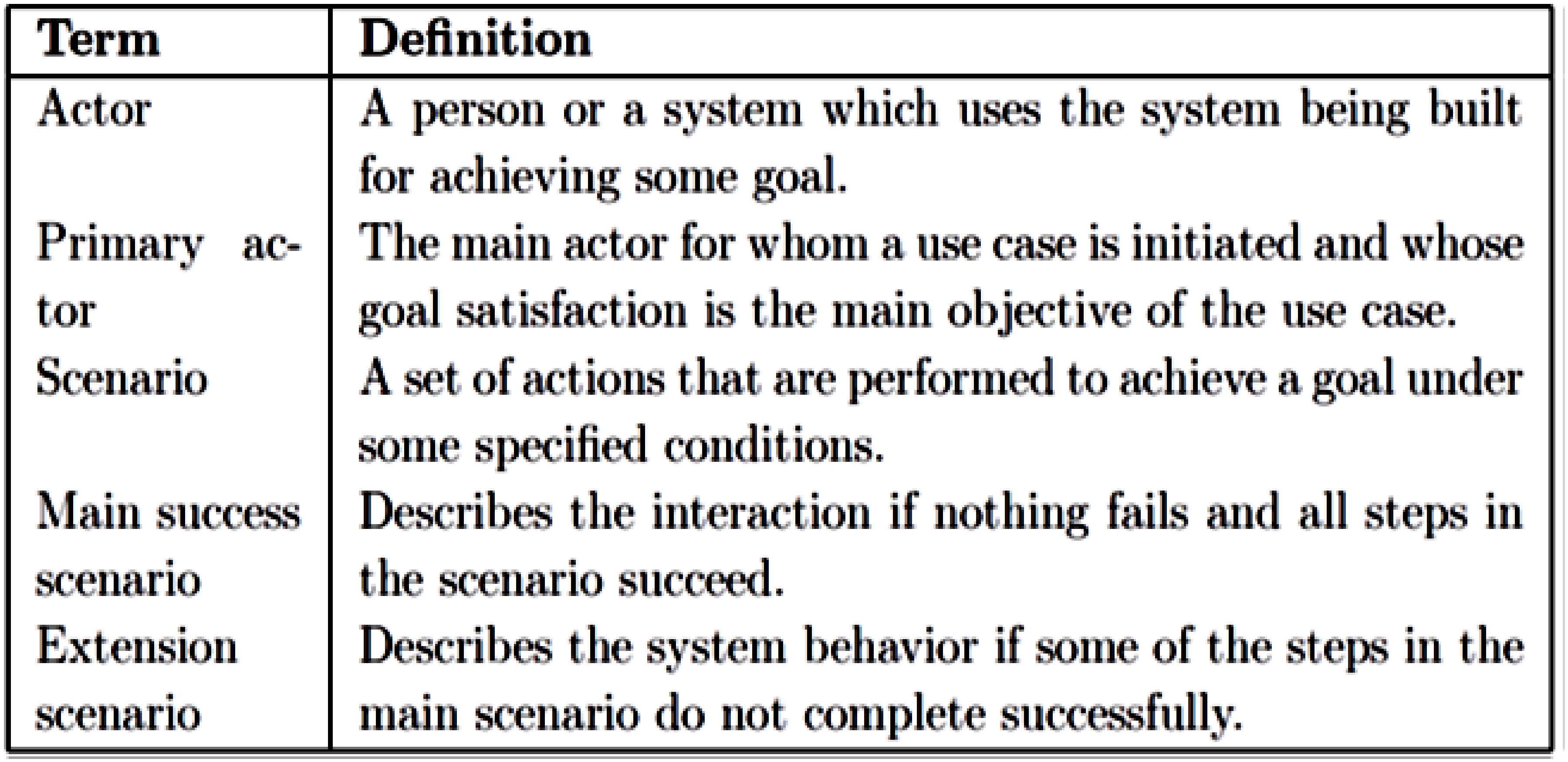


ATM machine example



Scenarios and Textual Use Cases

- Specify behavior of use case by description
- Examples include informal structured text, formal structured text with conditions, and pseudo code. Use case terms shown



Term	Definition
Actor	A person or a system which uses the system being built for achieving some goal.
Primary actor	The main actor for whom a use case is initiated and whose goal satisfaction is the main objective of the use case.
Scenario	A set of actions that are performed to achieve a goal under some specified conditions.
Main success scenario	Describes the interaction if nothing fails and all steps in the scenario succeed.
Extension scenario	Describes the system behavior if some of the steps in the main scenario do not complete successfully.

Textual Use Cases

- The use cases are generally numbered for reference purposes. The name of the use case specifies the goal of the primary actor
- The primary actor can be a person or a system. The primary actor can also be another software which might request a service
- The precondition of a use case specifies what the system will ensure before allowing the use case to be initiated
- Common preconditions are "***user is logged in***", "***input data exists in files or other data structures***", etc.

Textual Use Cases

- The use case description list contains some actions that are not necessarily tied to the goals of the primary actor
- The use case has to ensure that the goals of all stakeholders (including the system itself) are also satisfied

On-line Auction System

- On-line auction system is to be built for a university community, called the University Auction System (UAS), through which different members of the university can sell and buy goods
- We will assume that there is a separate financial subsystem through which the payments are made and that each buyer and seller has an account in it
- Consider the main use cases of this system—"***put an item for auction***", "***make a bid***", and "***complete an auction***"
- The use cases are self-explanatory.
- This is the great value of use cases—they are natural and story-like which makes them easy to understand by both an analyst and a layman

Use Case 1

– *UC1*: **Put an item for auction**

Primary Actor: Seller

Precondition: Seller has logged in

Main Success Scenario:

1. Seller posts an item (its category, description, picture, etc.) for auction
2. System shows past prices of similar items to seller
3. Seller specifies the starting bid price and a date when auction will close
4. System accepts the item and posts it

Exception Scenarios:

– 2 a) There are no past items of this category

- System tells the seller this situation

Use Case 2

– *UC2*: **Make a bid**

Primary Actor: Buyer

Precondition: The buyer has logged in

Main Success Scenario:

1. Buyer searches or browses and selects some item
2. System shows the rating of the seller, the starting bid, the current bids, and the highest bid; asks buyer to make a bid
3. Buyer specifies a bid price
4. System accepts the bid; Blocks funds in bidders account
5. System updates the max bid price, informs other users, and updates the records for the item

Exception Scenarios:

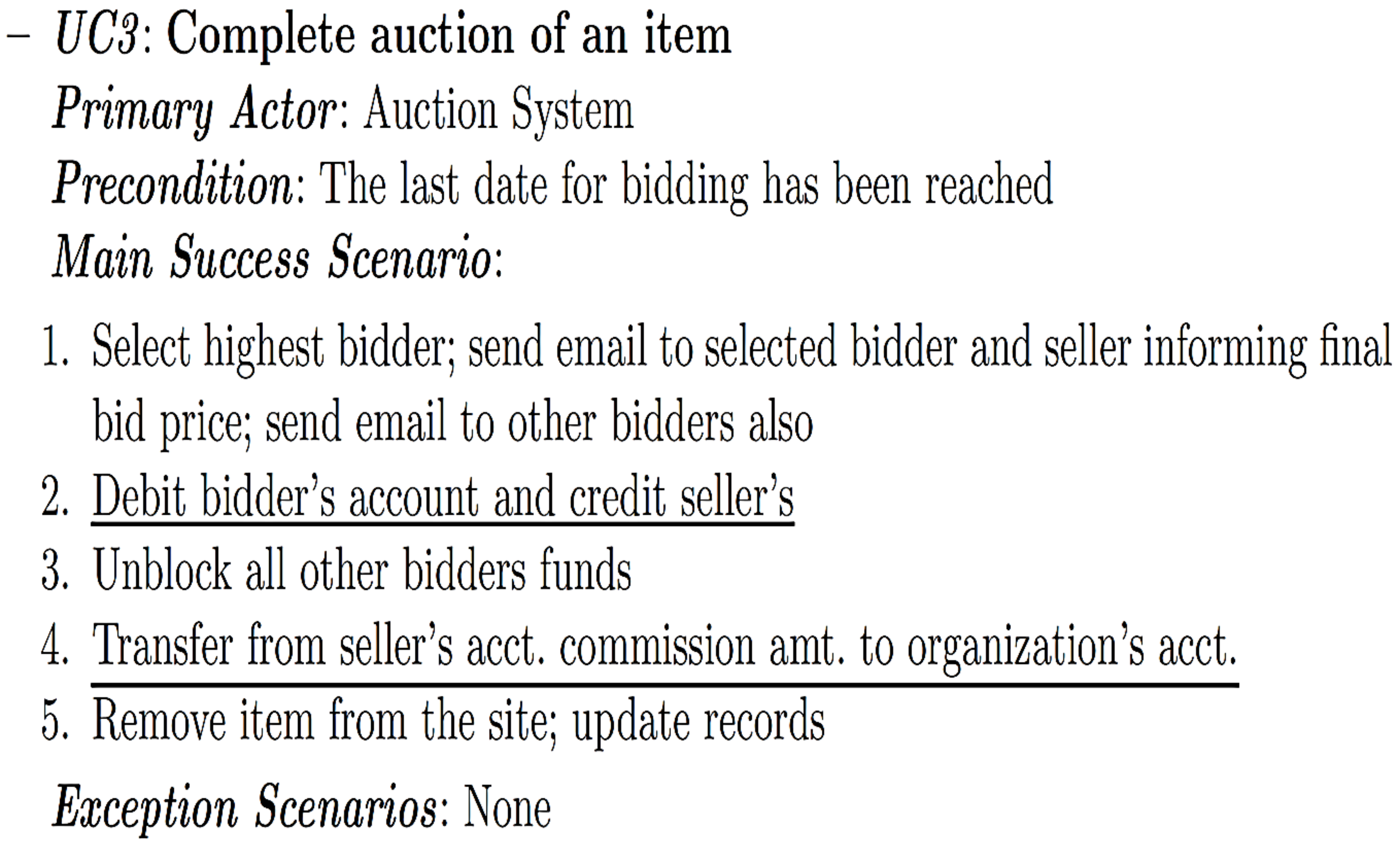
– 3 a) The bid price is lower than the current highest

- System informs the bidder and asks to rebid

– 4 a) The bidder does not have enough funds in his account

- System cancels the bid, asks the user to get more funds

Use Case 3



– *UC3*: Complete auction of an item

Primary Actor: Auction System

Precondition: The last date for bidding has been reached

Main Success Scenario:

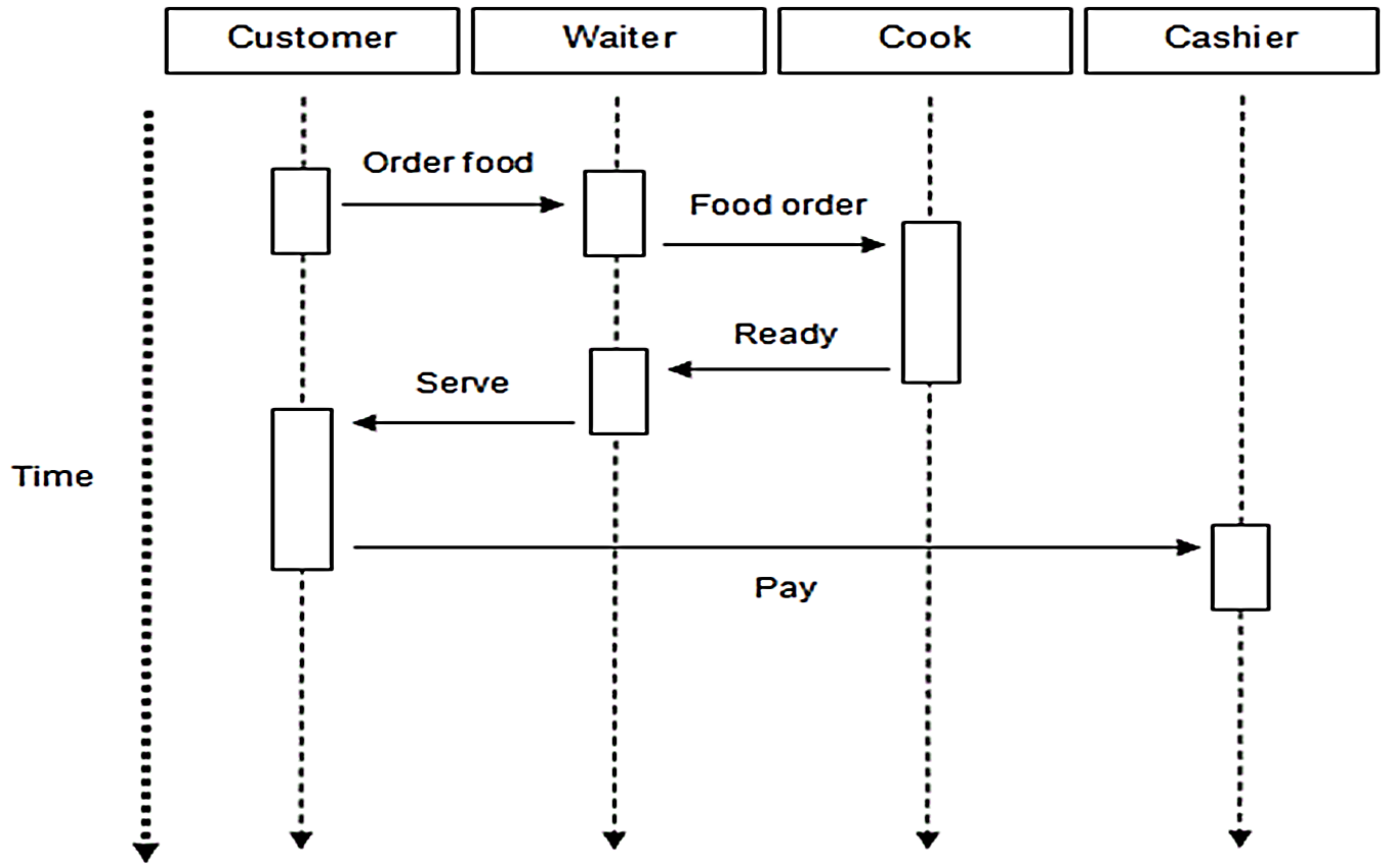
1. Select highest bidder; send email to selected bidder and seller informing final bid price; send email to other bidders also
2. Debit bidder's account and credit seller's
3. Unblock all other bidders funds
4. Transfer from seller's acct. commission amt. to organization's acct.
5. Remove item from the site; update records

Exception Scenarios: None

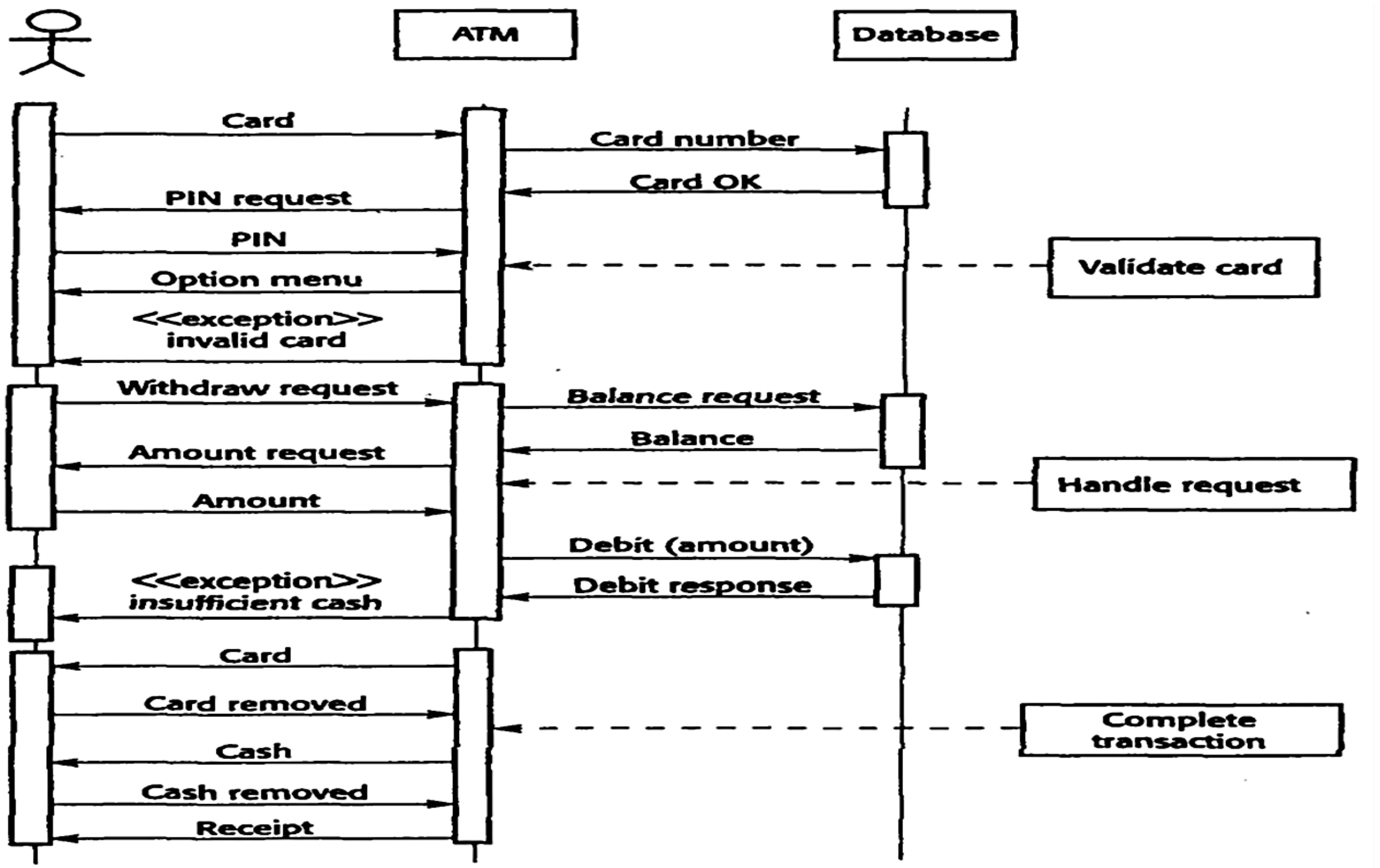
Sequence Diagrams

- Sequence diagram is an interaction diagram typically captures the behavior of a use case and models how the different objects in the system collaborate to implement the use case (dynamic behavior)
- When drawing a sequence diagram, **lifeline notation** elements are placed across the top of the diagram
- Lifelines are drawn as a box with a dashed line descending from the center of the bottom edge
- The lifeline's name is placed inside the box

Restaurant Sequence Diagram



ATM Machine Sequence Diagram



SRS Document

- An SRS document is generated as the output of requirements analysis
- The SRS should be a consistent, correct, unambiguous, and complete document
- The developer of the system can prepare an SRS after detailed communication with the customer
- The IEEE and U.S. Department of Defense have proposed a candidate format for representing the SRS
- The general outline of the SRS document is as follows:

Organization of an SRS Document

1. *Introduction*
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Definitions, Acronyms, and Abbreviations
    - 1.4 References
    - 1.5 Overview
2. *The Overall Description*
    - 2.1 Product Perspective
        - 2.1.1 System Interfaces
        - 2.1.2 Interfaces
        - 2.1.3 Hardware Interfaces
        - 2.1.4 Software Interfaces
        - 2.1.5 Communications Interfaces
        - 2.1.6 Memory Constraints
        - 2.1.7 Operations
        - 2.1.8 Site Adaptation Requirements
    - 2.2 Product Functions
    - 2.3 User Characteristics
    - 2.4 Constraints
    - 2.5 Operating Environment
    - 2.6 User Environment
    - 2.7 Assumptions and Dependencies
    - 2.8 Apportioning of Requirements

Organization of an SRS Document

3. *Specific Requirements*
  - 3.1 External Interfaces
    - 3.1.1 User Interface
    - 3.1.2 Hardware Interface
    - 3.1.3 Software Interface
    - 3.1.4 Communication Interface
  - 3.2 Functions
  - 3.3 Performance Requirements
  - 3.4 Logical Database Requirements
  - 3.5 Design Constraints
    - 3.5.1 Standards Compliance
  - 3.6 Software System Attribute
    - 3.6.1 Reliability
    - 3.6.2 Availability
    - 3.6.3 Security
    - 3.6.4 Maintainability
    - 3.6.5 Portability
  - 3.7 Organizing the Specific Requirements
    - 3.7.1 System Mode
    - 3.7.2 User Class
    - 3.7.3 Objects
    - 3.7.4 Feature
    - 3.7.5 Stimulus
    - 3.7.6 Response
    - 3.7.7 Functional Hierarchy
  - 3.8 Additional Comments

4. *Supporting Information*
  - 4.1 Table of Contents and Index
  - 4.2 Appendixes

Benefits of a SRS

- Establish the basis for agreement between the customers and the suppliers on what the software product is to do
- Reduce the development effort
- Provide a basis for estimating costs and schedules
- Provide a baseline for validation and verification
- Facilitate transfer
- Serves as a basis for enhancement