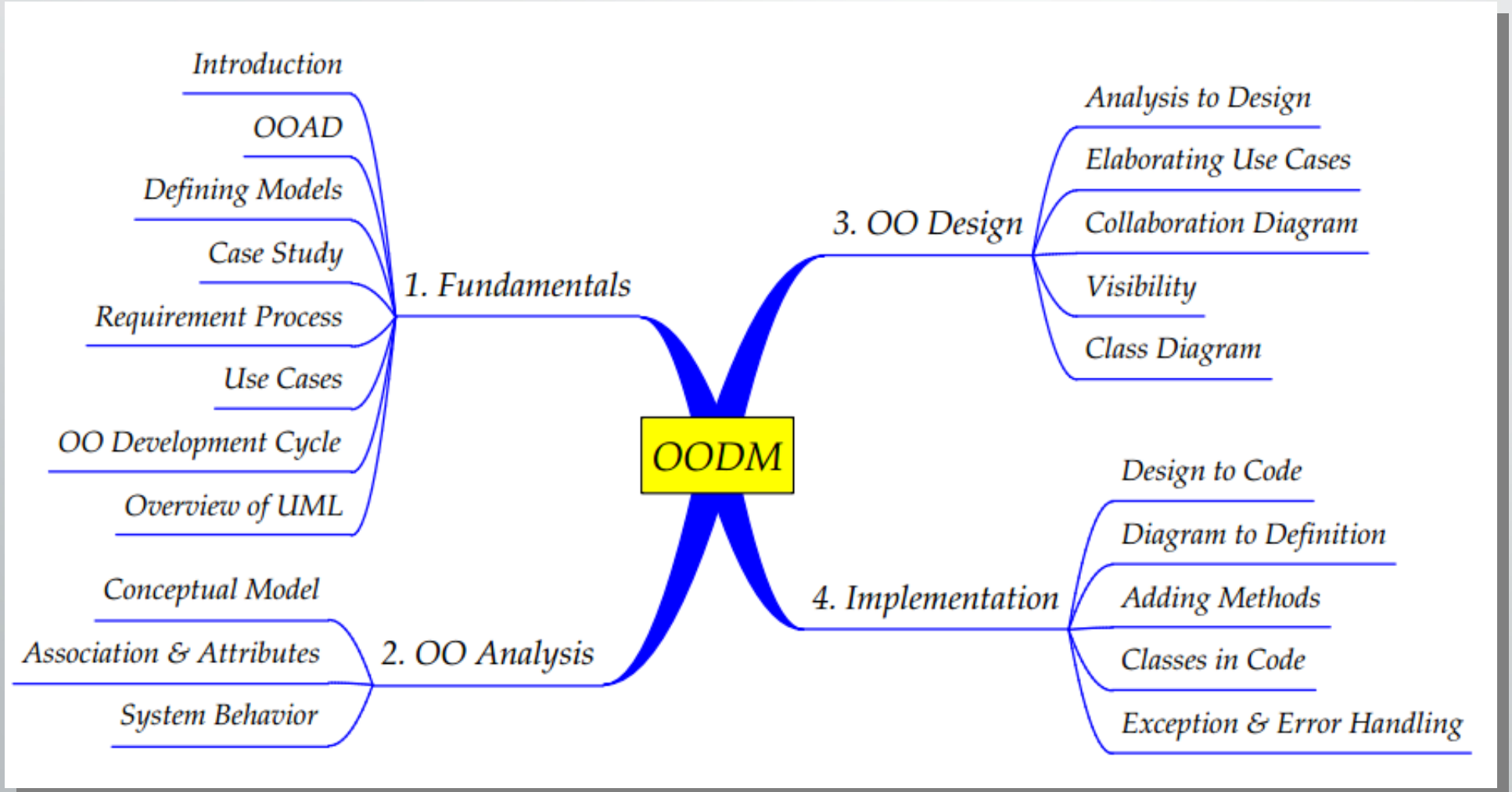
Object Oriented Design and Modeling through UML

[BE IT-6th Semester]

Rishi K. Marseni
{rishimarseni@gmail.com}

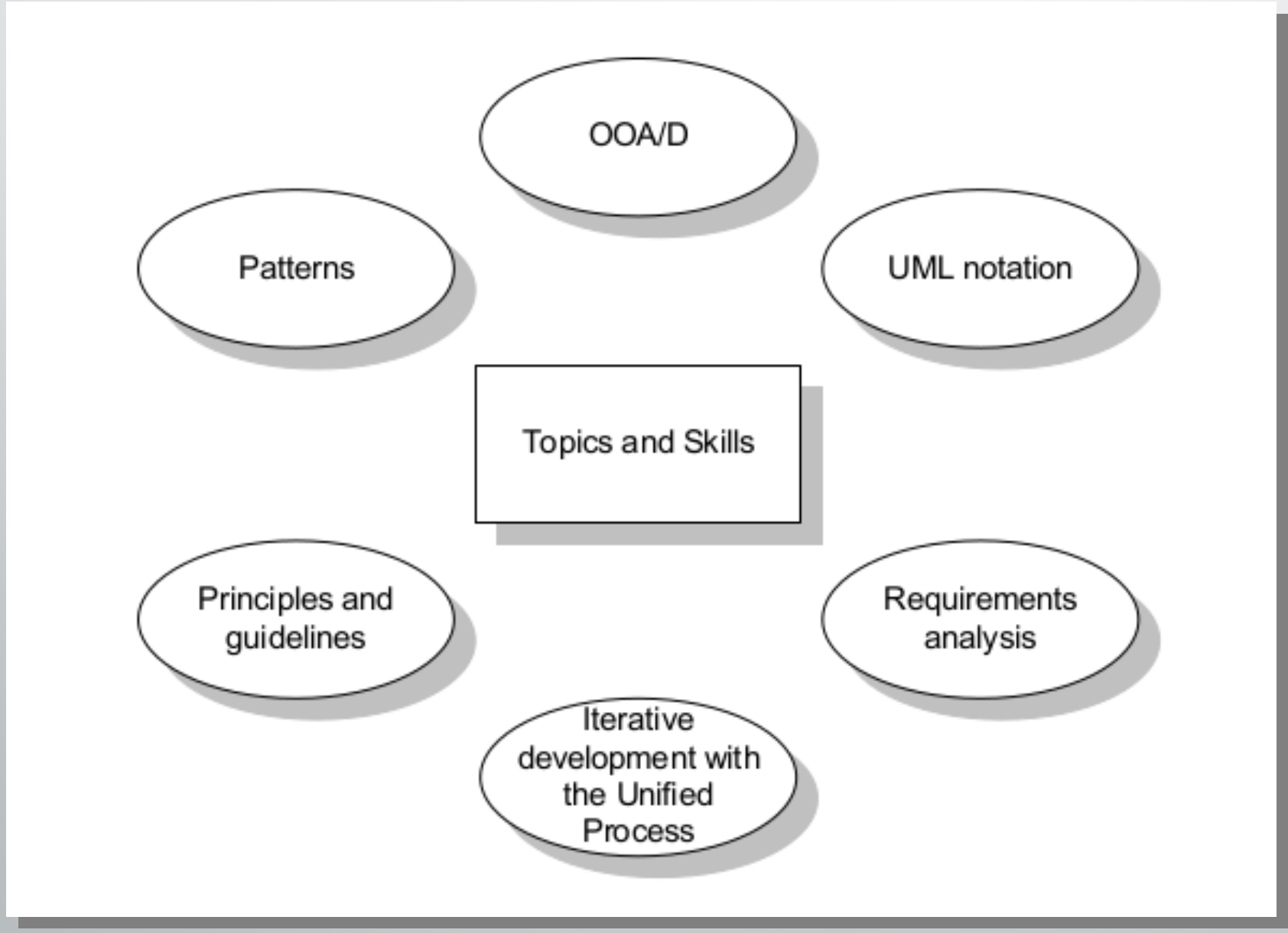
Course Content – Abstract



Text Book: **Applying UML and Patterns** by **Craig Larman**

Reference Book: **Software Engineering** by **Ian Sommerville**

Course objectives



1. OO Fundamentals

1.1. Introduction

1.2. OOAD

1.3. Defining Models

1.4. Case Study

1.5. Requirement Process

1.6. Use Cases

1.7. OO Development Cycle

1.8. Overview of UML

Requirements engineering

- The process of establishing the services that a customer requires from a system and the constraints under which it operates and is developed.
- The system requirements specifications(SRS) are the descriptions of the system services and constraints that are generated during the requirements engineering process.

What is a requirement?

It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

This is inevitable as requirements may serve a dual function

- May be the basis for a bid for a contract - therefore must be open to interpretation;
- May be the basis for the contract itself - therefore must be defined in detail;
- Both these statements may be called requirements.

Requirements abstraction (Davis)

“If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently abstract way that a solution is not pre-defined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization’s needs. Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called the requirements document for the system.”

Understanding Requirements (The FURPS+)

In the UP, requirements are categorized according to the FURPS+ model [Grady92], a useful mnemonic with the following meaning:

Functional—features, capabilities, security.

Usability—human factors, help, documentation.

Reliability—frequency of failure, recoverability, predictability.

Performance—response times, throughput, accuracy, availability, resource usage.

Supportability—adaptability, maintainability, internationalization, configure-ability.

It is helpful to use FURPS+ categories (or some categorization scheme) as a checklist for requirements coverage, to reduce the risk of not considering some important facet of the system.

Understanding Requirements (The FURPS+)

The "+" in FURPS+ indicates ancillary and sub-factors, such as:

- Implementation—resource limitations, languages and tools, hardware, ...
- Interface—constraints imposed by interfacing with external systems.
- Operations—system management in its operational setting.
- Packaging
- Legal—licensing and so forth.

Some of these requirements are collectively called the quality attributes, quality requirements, or the "-ilities" of a system. These include usability, reliability, performance, and supportability. In common usage, requirements are categorized as functional (behavioral) or non-functional (everything else).

System stakeholders

Any person or organization who is affected by the system in some way and so who has a legitimate interest

Stakeholder types

- End users
- System managers
- System owners
- External stakeholders

Stakeholders in the Mentcare system

- Patients whose information is recorded in the system.
- Doctors who are responsible for assessing and treating patients.
- Nurses who coordinate the consultations with doctors and administer some treatments.
- Medical receptionists who manage patients' appointments.
- IT staff who are responsible for installing and maintaining the system.
- A medical ethics manager who must ensure that the system meets current ethical guidelines for patient care.
- Health care managers who obtain management information from the system.
- Medical records staff who are responsible for ensuring that system information can be maintained and preserved, and that record keeping procedures have been properly implemented.

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.

Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

Requirements completeness and consistency

In principle, requirements should be both complete and consistent.

Complete =>They should include descriptions of all facilities required.

Consistent=> There should be no conflicts or contradictions in the descriptions of the system facilities.

Mentcare system: functional requirements

- A user shall be able to search the appointments lists for all clinics.
- The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
- Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Requirements imprecision: SE problem

Problems arise when functional requirements are not precisely stated.

Ambiguous requirements may be interpreted in different ways by developers and users.

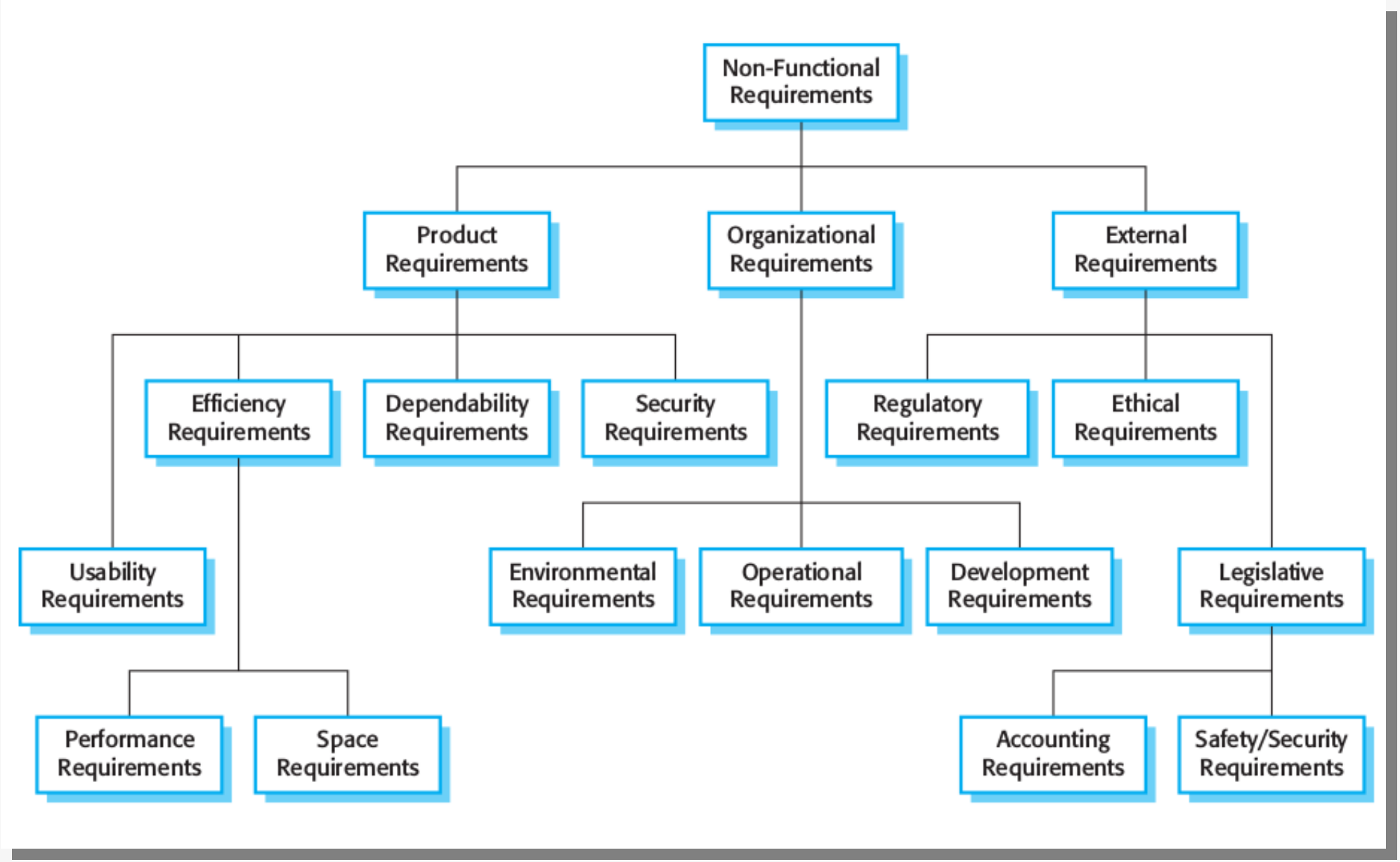
Consider the term ‘search’ in requirement 1

- User intention – search for a patient name across all appointments in all clinics;
- Developer interpretation – search for a patient name in an individual clinic. User chooses clinic then search.

Non-functional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements.
- Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements.
- If these are not met, the system may be useless.

Types of nonfunctional requirement



Non-functional classifications

Product requirements

Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.

Organizational requirements

Requirements which are a consequence of organizational policies and procedures e.g. process standards used, implementation requirements, etc.

External requirements

Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Examples of nonfunctional requirements in the Mentcare system

Product requirement
The Mentcare system shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

Organizational requirement
Users of the Mentcare system shall authenticate themselves using their health authority identity card.

External requirement
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Goals and requirements

Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.

Goal

- A general intention of the user such as ease of use.

Verifiable non-functional requirement

- A statement using some measure that can be objectively tested.

Goals are helpful to developers as they convey the intentions of the system users.

Metrics for specifying nonfunctional requirements

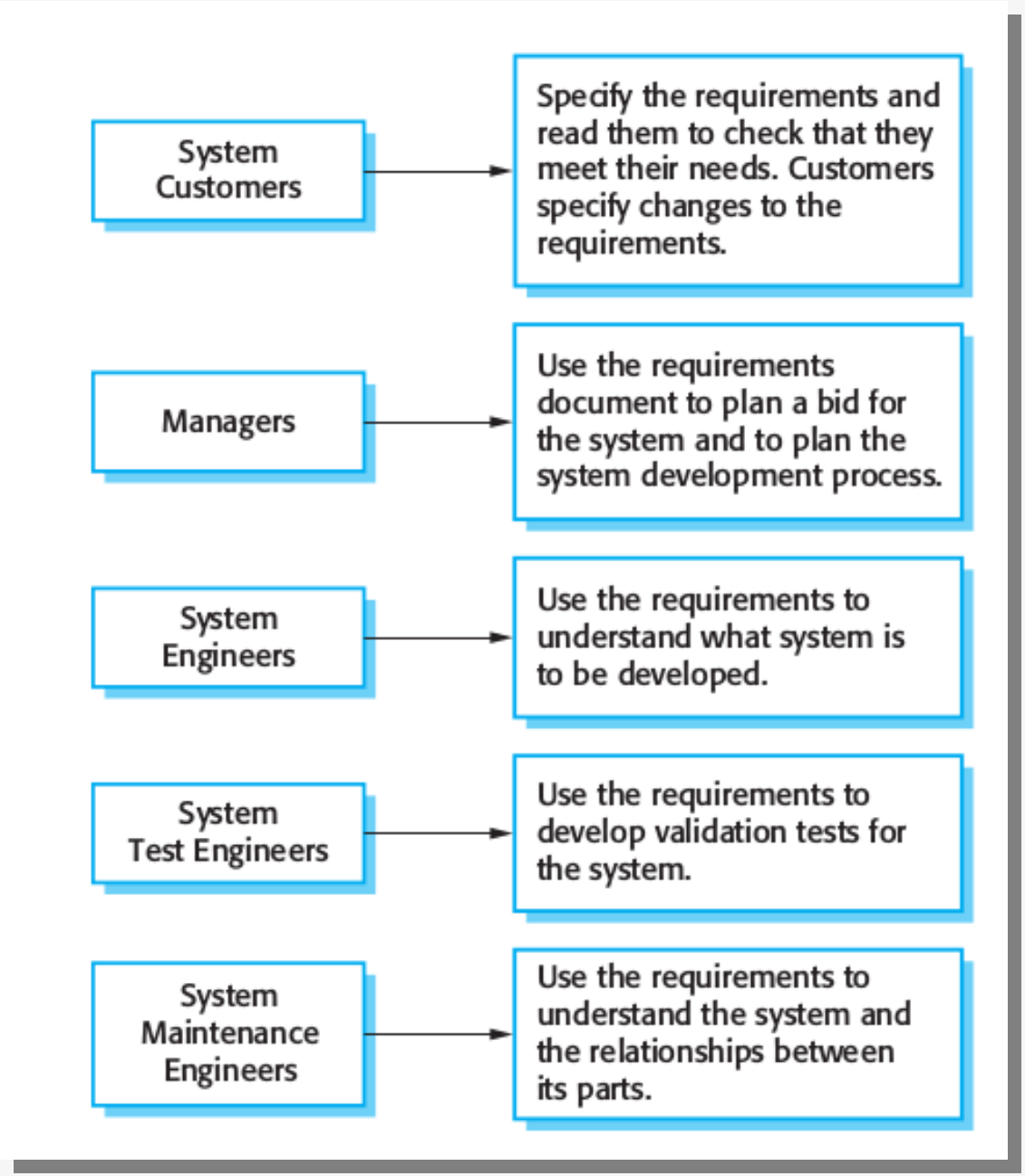
Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Figure: Metrics for specifying non-functional requirements

The software requirements document

- The software requirements document is also called the software requirements specification or SRS.
- It is an official statement of what the system developers should implement.
- It should include both user requirements and system requirements.
- The requirements document has a diverse set of users, ranging from the senior management of the organization that is paying for the system to the engineers responsible for developing the software
- The software requirements document is the official statement of what is required of the system developers
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. it should set of WHAT the system should do rather than HOW it should do it..

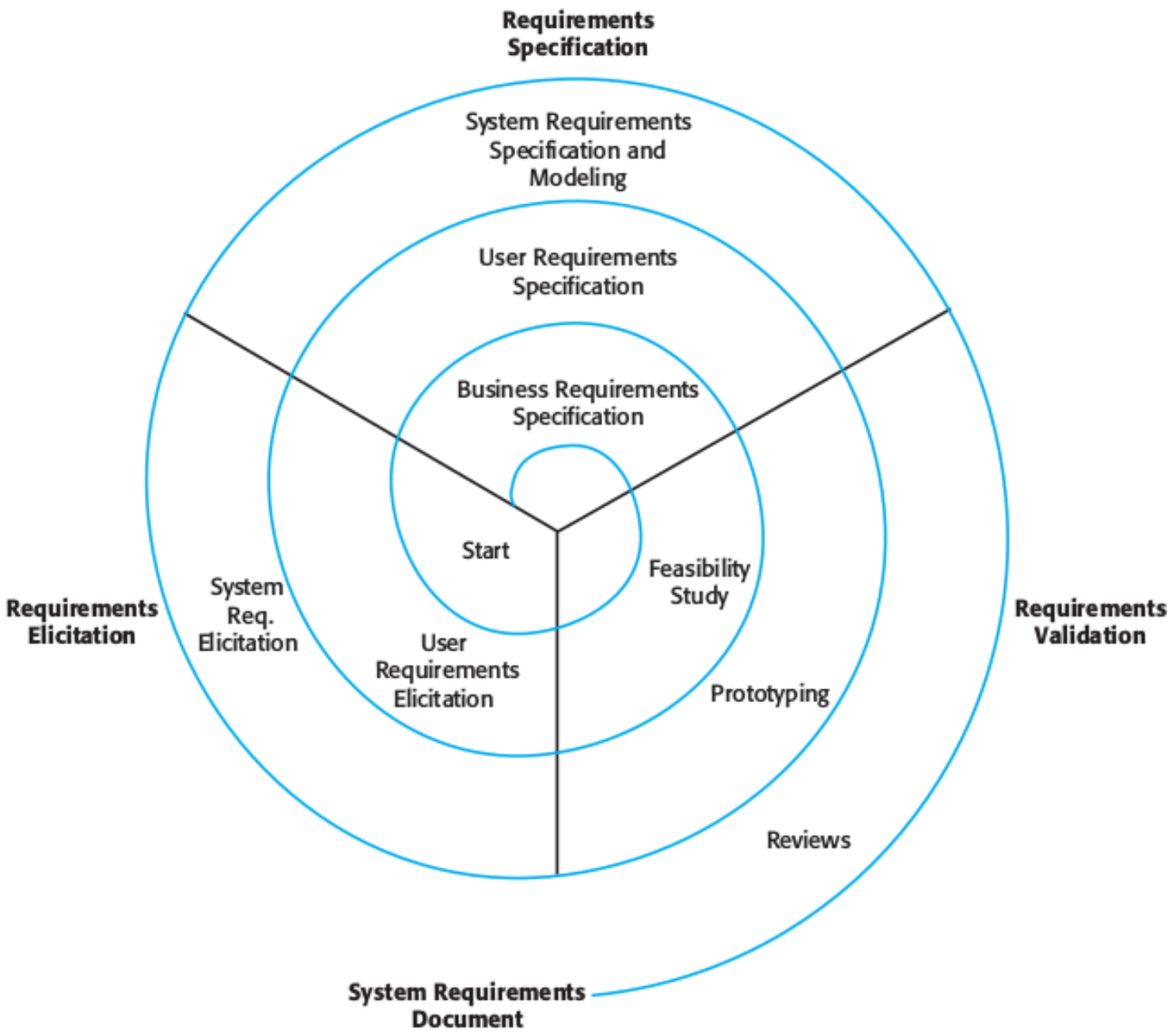
Users of a requirements document



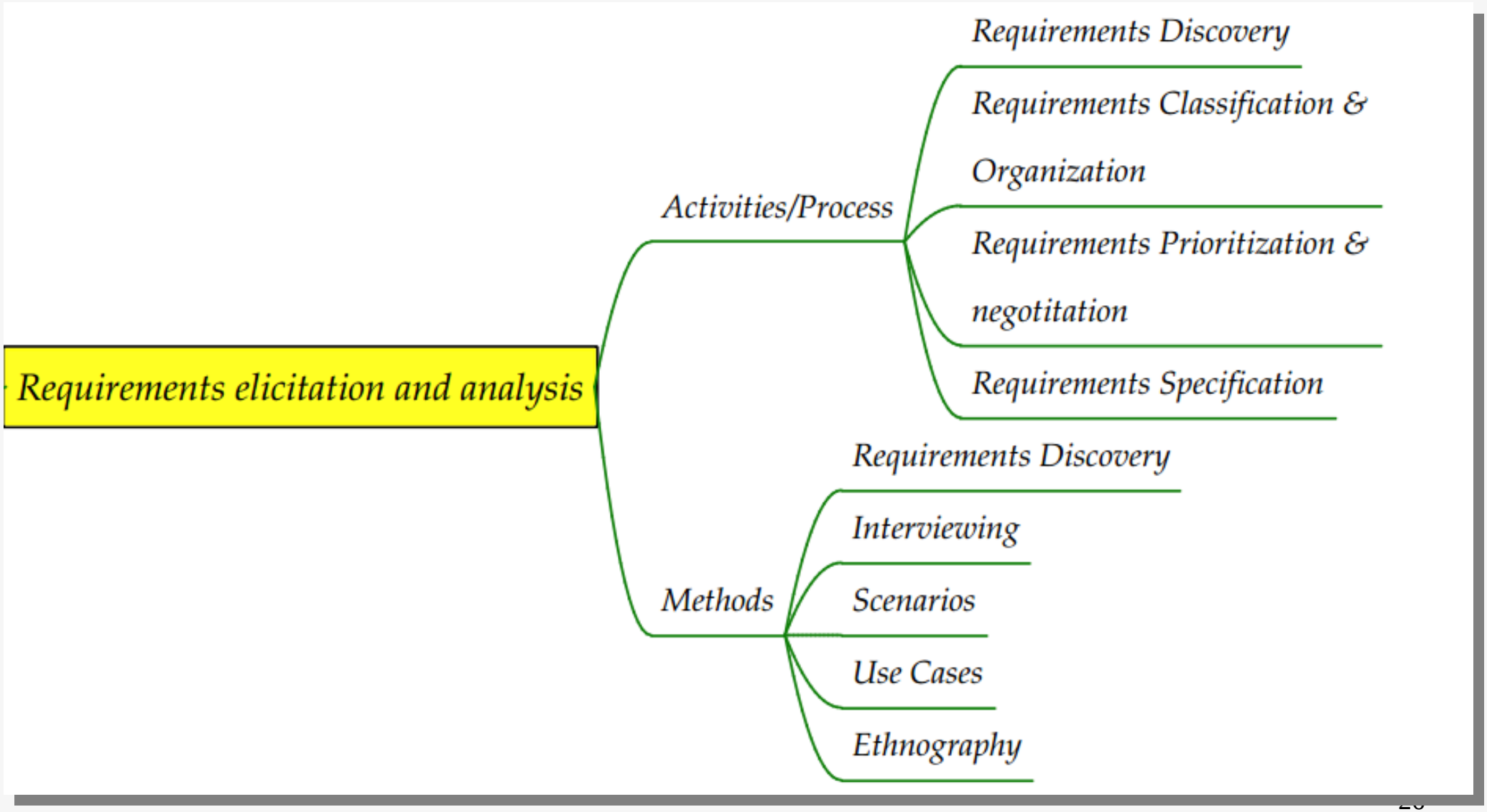
Requirements engineering processes

there are a number of generic activities common to all processes

- Requirements elicitation;
- Requirements analysis;
- Requirements validation;
- Requirements management.


Requirements Specification
System Requirements Specification and Modeling
User Requirements Specification
Business Requirements Specification
Start
Feasibility Study
Requirements Elicitation
System Req. Elicitation
User Requirements Elicitation
Prototyping
Requirements Validation
Reviews
System Requirements Document

Requirement Elicitation and Analysis



Requirement Elicitation

- Requirements elicitation is a requirements discovery process.
- Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.

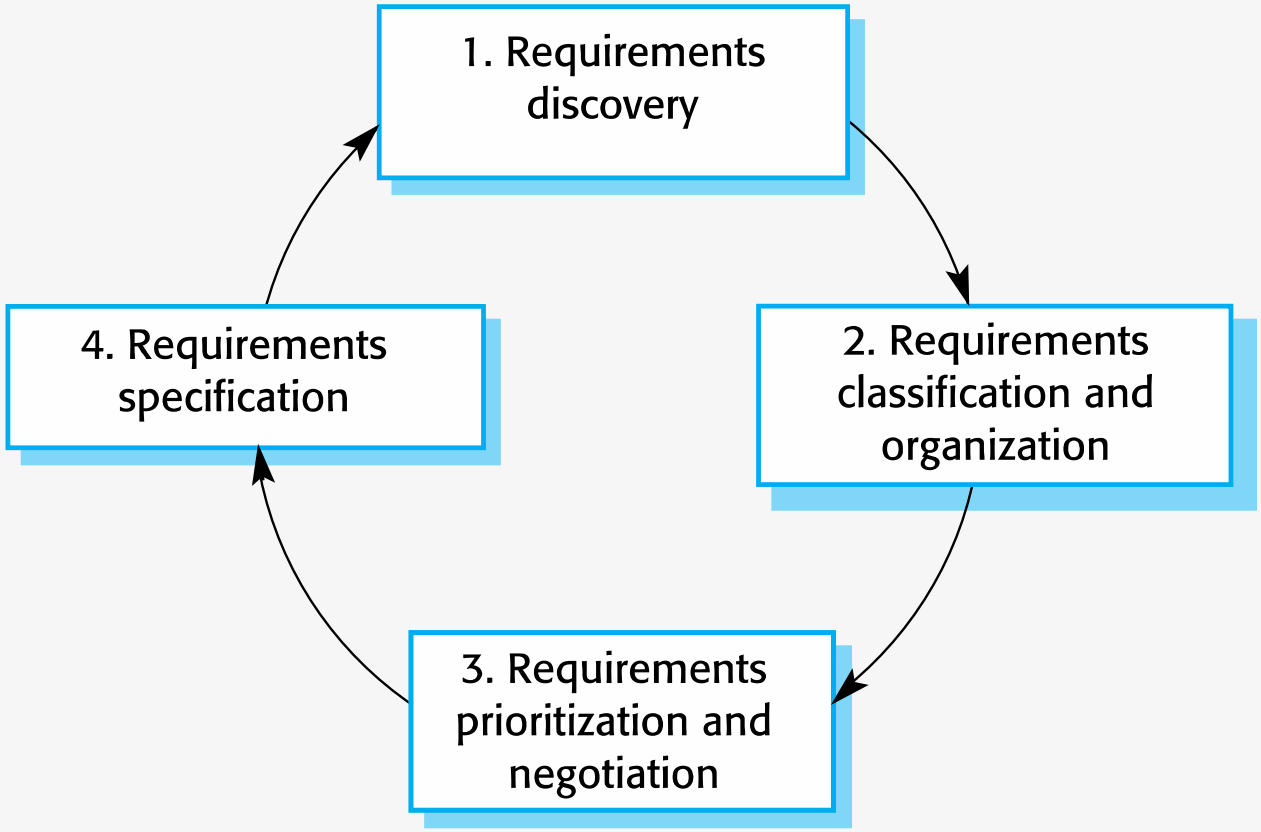
Stages include:

- Requirements discovery,
- Requirements classification and organization,
- Requirements prioritization and negotiation,
- Requirements specification.

Problems of requirements elicitation

- Stakeholders don’t know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organizational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment may change.

The requirements elicitation and analysis process



Requirements discovery

The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.

Interaction is with system stakeholders from managers to external regulators.

Systems normally have a range of stakeholders.

Requirement discovery methods:

- Interview
- Questionnaire
- survey

Interviewing

Types of interview

- Closed interviews based on per-determined list of questions
- Open interviews where various issues are explored with stakeholders.

Effective interviewing

- Be open-minded
- avoid per-conceived ideas about the requirements
- willing to listen to stakeholders.
- Prompt the user to talk by giving them some requirement idea.

Problems with interviews

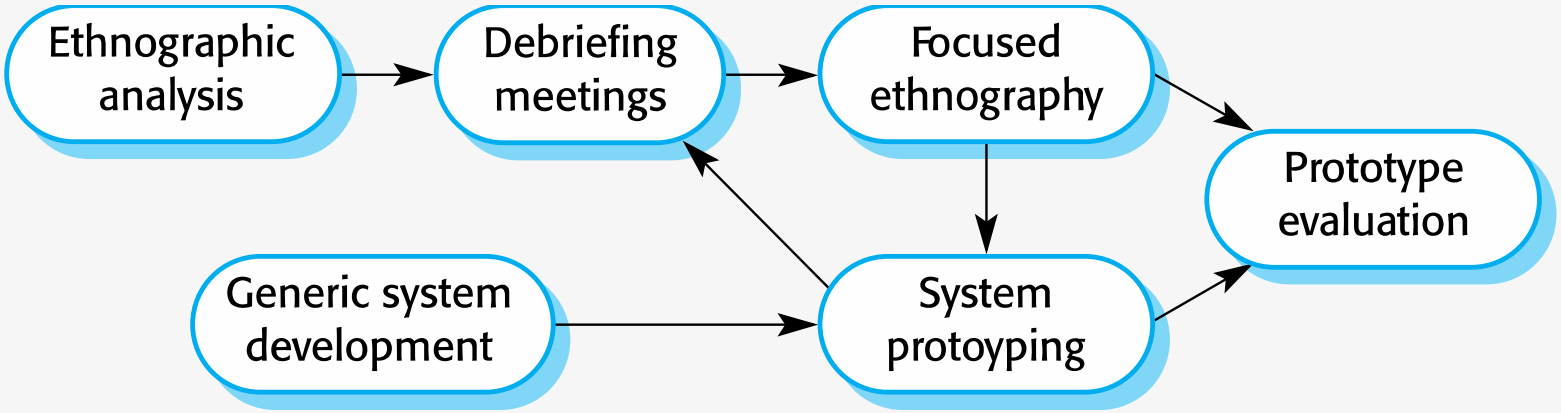
Application specialists may use language to describe their work that isn’t easy for the requirements engineer to understand.

- Interviews are not good for understanding domain requirements
- Requirements engineers cannot understand specific domain terminology
- Users may not be technically so strong to tell accurate requirement.

Ethnography

- A social scientist spends a considerable time observing and analyzing how people actually work.
- Social and organizational factors of importance may be observed.
- Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.
- Combines ethnography with prototyping
- Prototype development results in unanswered questions which focus the ethnographic analysis.
- The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant.

Ethnography and prototyping for requirements analysis



Stories and scenarios

- Scenarios and user stories are real-life examples of how a system can be used.
- Stories and scenarios are a description of how a system may be used for a particular task.
- Because they are based on a practical situation, stakeholders can relate to them and can comment on their situation with respect to the story.

Scenarios

A structured form of user story

Scenarios should include

A description of the starting situation;

A description of the normal flow of events;

A description of what can go wrong;

Information about other concurrent activities;

A description of the state when the scenario finishes.

Use cases

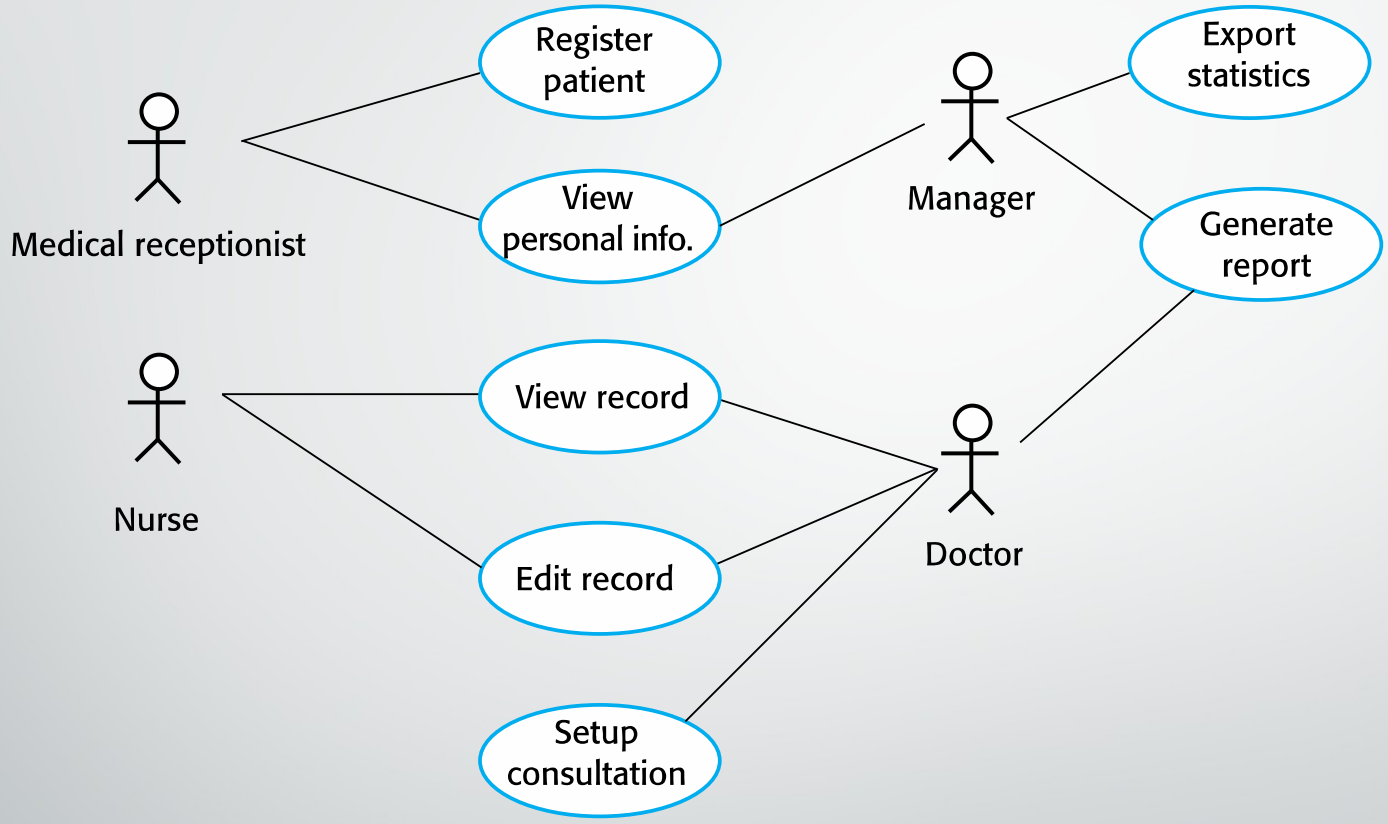
Use-cases are a kind of scenario that are included in the UML.

Use cases identify the actors in an interaction and which describe the interaction itself.

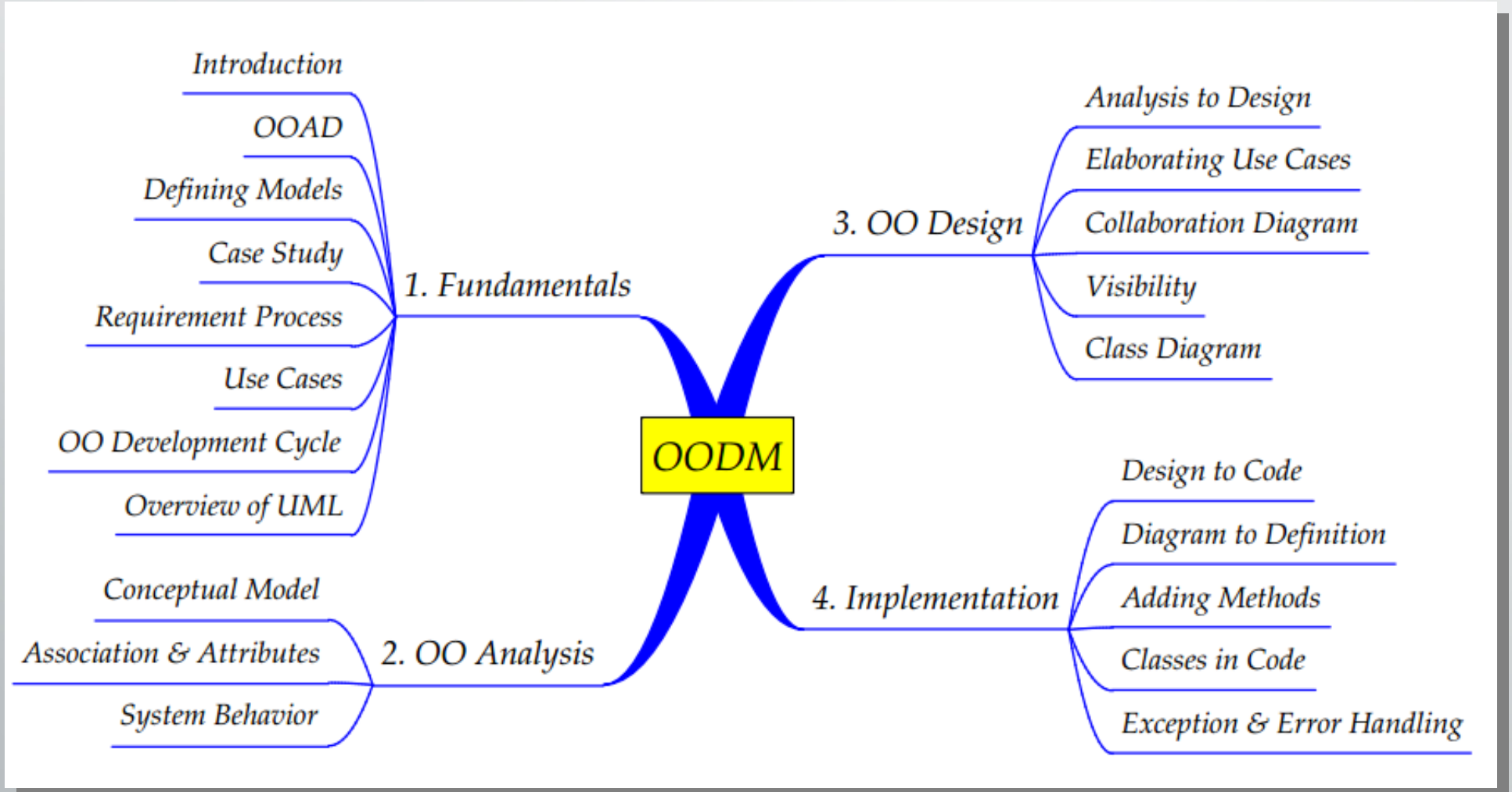
A set of use cases should describe all possible interactions with the system.

UML sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.

Use cases for the Mentcare system

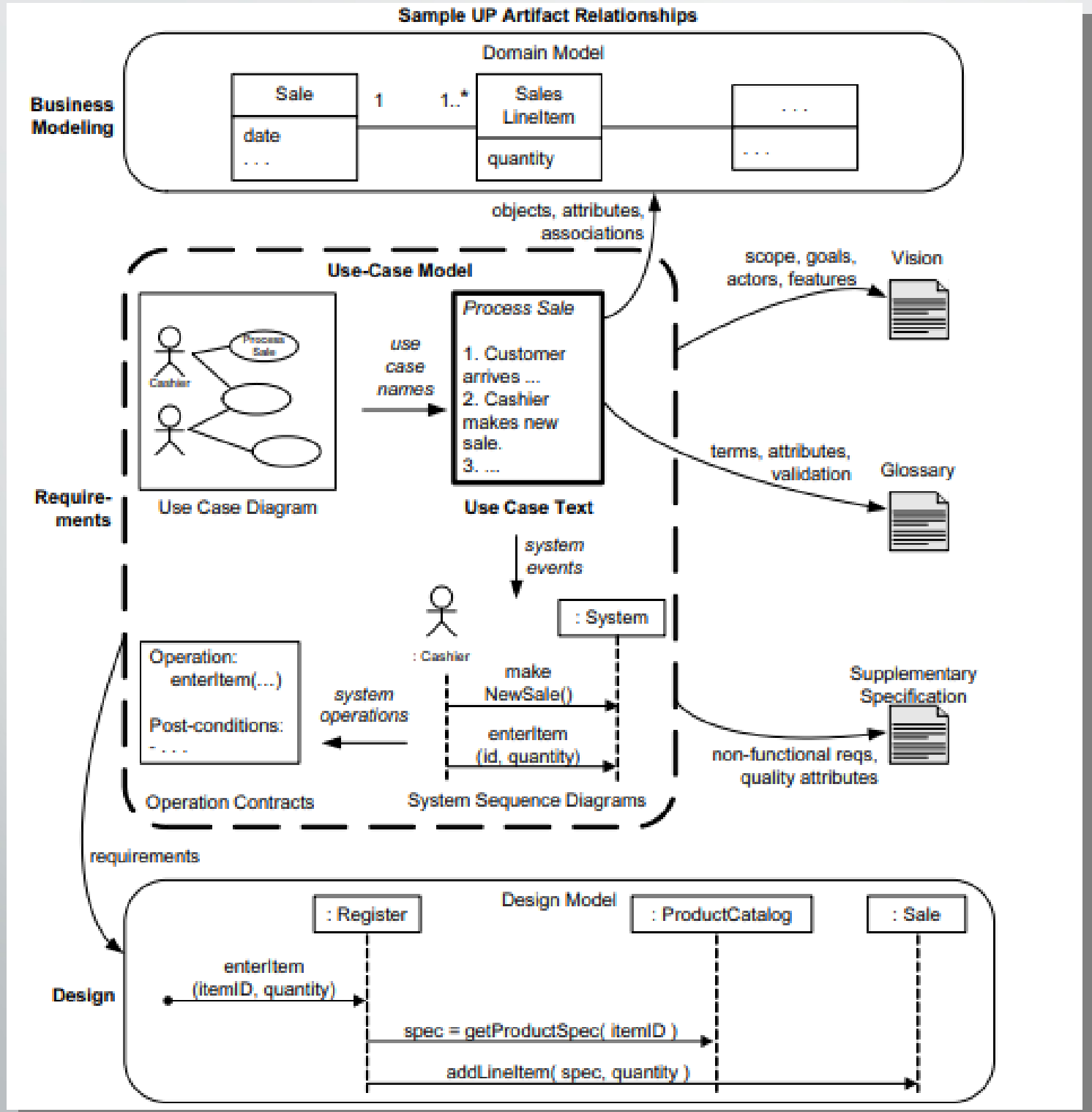


Course Content – Abstract



Text Book: **Applying UML and Patterns** by **Craig Larman**

Reference Book: **Software Engineering** by **Ian Sommerville**


Sample UP Artifact Relationships
Business Modeling
Domain Model
Sale
date
. . .
1
1..*
Sales LineItem
quantity
. . .
. . .
objects, attributes, associations
Use-Case Model
scope, goals, actors, features
Vision
Process Sale
Cashier
use case names
Process Sale
1. Customer arrives ...
2. Cashier makes new sale.
3. ...
Require-ments
Use Case Diagram
Use Case Text
terms, attributes, validation
Glossary
system events
: Cashier
: System
Operation:
enterItem(...)
Post-conditions:
- . . .
system operations
make NewSale()
enterItem (id, quantity)
Supplementary Specification
non-functional reqs, quality attributes
Operation Contracts
System Sequence Diagrams
requirements
Design Model
: Register
: ProductCatalog
: Sale
Design
enterItem (itemID, quantity)
spec = getProductSpec(itemID)
addLineItem(spec, quantity)

Use Cases

- Use cases are text stories, widely used to discover and record requirements
- Use cases are text documents,not diagrams
- Use-case modeling is primarily an act of writing text, not drawing diagrams
- There is nothing object-oriented about use cases; we’re not doing OO analysis when writing them
- Use cases are a key requirements input to classic OOA/D.
- UML use case diagram to show the names of use cases and actors, and their relationships.

Use Case

- A use case is a text story
- Widely used to discover and record (mostly functional) requirements
- Some actor(s) using a system to meet specific goals
- Use cases are used for answering questions:

  → *Who is using the system, what are their typical scenarios of use, and what are their goals?*

Use Cases

- **Actor**: entity that shows a behavior,

  → e.g. a person (role), computer system, or organization

- **Scenario**: specific sequence of actions and interactions between actors and a system

  → use case instance

  → singe path of using the system

  → e.g., purchasing some items with cash

- **Use case**: collection of related success & failure scenarios that describe an actor using a system to support a goal

Use Case: An Example

- UC Handle Returns:

  Main success Scenario: A customer arrives at a checkout with items to return. The cashier uses the POS system to record each returned item …

  Alternate Scenarios:

  if the customer paid by credit ...

  If the item identifier is not found in the system …

  If the system detects failure to communicate with the external accounting system …

Use Case Model

- Set of all written use cases
- Model of the system’s functionality and environment
- Unified Process (UP) defined artifact within the requirements discipline
- UP also requires glossary.
- May optionally include a UML use case diagram
  - → use cases, actors, and their relationships
  - → context diagram

Three Kinds of Actors

- **Primary actor**

  → has user goals fulfilled through using services of the system under discussion

  → drives the use cases

- **Supporting actor**

  → provides a service to the system under discussion

  → e.g., payment authorization service

  → implies: clarification of external interfaces and protocols needed

- **Offstage actor**

  → has an interest in the behavior of the use case, but is not primary or supporting

  → e.g., a government tax agency

Use Case Formats

1. **Brief —** Terse one-paragraph summary, usually of the main success scenario. Useful during early requirements analysis, to get a quick sense of subject and scope. May take only a few minutes to create. The Process Sale example was brief.

- **Process Sale**: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items

Use Case Formats

2. Casual — Informal paragraph format.

Multiple paragraphs that cover various scenarios.

The prior ***Handle Returns***[44] example was casual.

3. Fully Dressed — the most elaborate.

All steps and variations are written in detail, and there are supporting sections, such as preconditions and success guarantees.

These detailed use cases are written after many uses cases have been identified and written in a brief format

Fully Dressed Use Case Template

Use Case Section	Comment
Use Case Name	Start with a verb.
Scope	The system under design.
Level	"user-goal" or "subfunction"
Primary Actor	Calls on the system to deliver its services.
Stakeholders and Interests	Who cares about this use case, and what do they want?
Preconditions	What must be true on start, *and* worth telling the reader?
Success Guarantee	What must be true on successful completion, *and* worth telling the reader.
Main Success Scenario	A typical, unconditional happy path scenario of success.
Extensions	Alternate scenarios of success or failure.
Special Requirements	Related non-functional requirements.
Technology and Data Variations List	Varying I/O methods and data formats.
Frequency of Occurrence	Influences investigation, testing, and timing of implementation.
Miscellaneous	Such as open issues.

Fully Dressed use case example

Use Case UC1: Process Sale

Scope: NextGen POS application

Level: user goal

Primary Actor: Cashier

Stakeholders and Interests:

→ Cashier: Wants accurate, fast entry, and no payment errors, as cash drawer shortages are deducted from his/her salary.

→ Company: Wants to accurately record transactions and satisfy customer interests. Wants to ensure that Payment Authorization Service payment receivables are recorded. Wants some fault tolerance to allow sales capture even if server components (e.g., remote credit validation) are unavailable. Wants automatic and fast update of accounting and inventory.

Fully Dressed use case example

Stakeholders and Interests(contd.):

→ Customer: Wants purchase and fast service with minimal effort.

→ Salesperson: Wants sales commissions updated.

→ Government Tax Agencies: Want to collect tax from every sale. May be multiple agencies, such as national, state, and county.

→ Payment Authorization Service: Wants to receive digital authorization requests in the correct format and protocol. Wants to accurately account for their payables to the store.

Preconditions: Cashier is identified and authenticated.

Success Guarantee (or Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

Fully Dressed use case example

Main Success Scenario (or Basic Flow) :

1. Customer arrives at POS checkout with goods and/or services to purchase.

2. Cashier starts a new sale.

3. Cashier enters item identifier.

4. System records sale lineitem and presents item desc., price, & running total.

Price calculated from a set of price rules. Cashier repeats 3-4 until indicates done.

5. System presents total with taxes calculated.

6. Cashier tells Customer the total, and asks for payment.

7. Customer pays and System handles payment.

8. System logs completed sale and sends sale and payment information to the external

Accounting system (for accounting and commissions) and Inventory system .

9. System presents receipt.

10. Customer leaves with receipt and goods (if any).

Fully Dressed use case example

Extensions (or Alternative Flows):

*a. At any time, System fails:

To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.

1. Cashier restarts System, logs in, and requests recovery of prior state.

2. System reconstructs prior state.

  2a. System detects anomalies preventing recovery:

    1. System signals error to the Cashier, records the error, and enters a clean state.

    2. Cashier starts a new sale.

3a. Invalid identifier:

    1 . System signals error and rejects entry.

3b. There are multiple of same ……

Fully Dressed use case example

Special Requirements:

- Touch screen UI on a large fiat panel monitor. Text must be visible from 1 meter.
- Credit authorization response within 30 seconds 90% of the time.
- Language internationalization on the text displayed.

Technology and Data Variations List:

3a. Item identifier entered by bar code laser scanner (if bar code is present) or keyboard.

7a. Credit account information entered by card reader or keyboard.

Frequency of Occurrence: Could be nearly continuous.

Open Issues:

- What are the tax law variations?
- Explore the remote service recovery issue.
- What customization is needed for different businesses?
- Must a cashier take their cash drawer when they log out?
- Can the customer directly use the card reader, or does the cashier have to do it?

A Two-Column Variation

Actor Action	**System Response**
Numbered action of the actor	*Numbered description of system response*

Alternative Course of events describes important alternatives or exceptions that may arise with respect to typical course.

A Two-Column Variation

Actor Action	System Response
1. The use case begins when a Customer arrives at a POST checkout with Items to purchase	
2. The Cashier records the identifier from each item	3. Determines the item price and adds the item information to the running sales transaction.
If there are more than one of the same type of item, the cashier can enter the quantity as well.	The description and price of the current item are presented.
4. On completion of item entry the Cashier indicates to the POST that item entry is complete.	5. Calculates and presents the sales total.

A Two-Column Variation

6. The Cashier tells the Customer the Total 7. The customer gives cash payment- the "cash tendered "- possibly greater than the sale total. 8. The Cashier records the cash received amount	9. Shows the balance due back to the Customer. Generates a receipt…
10. The Cashier deposits the cash received and extract the balance owing 12. The customer leaves with the items purchased	11. Logs the completed sale.

Guideline for writing use cases

1. Choose the system boundary:

Is it just a software application, the hardware and application as a unit, that plus a person using it, or an entire organization?

2. Identify the primary actors:

Those that have user goals fulfilled through using services of the system.

3. For each, identify their user goals:

Raise them to the highest user goal level that satisfies the EBP guideline.

4. Define use cases that satisfy user goals:

name them according to their goal. Usually, user goal-level use cases will be one-to-one with user goals.

Finding User goals

Actor	Goal
Cashier:	process sales, process rentals, handle return, cash in , cash out
Manager:	start up , shut down
System Admin:	add users, modify users, delete users, manage security, manage system
Sales Activity Sys.:	analyze sales and performance data
Customer:	Buy Items, Refund Items

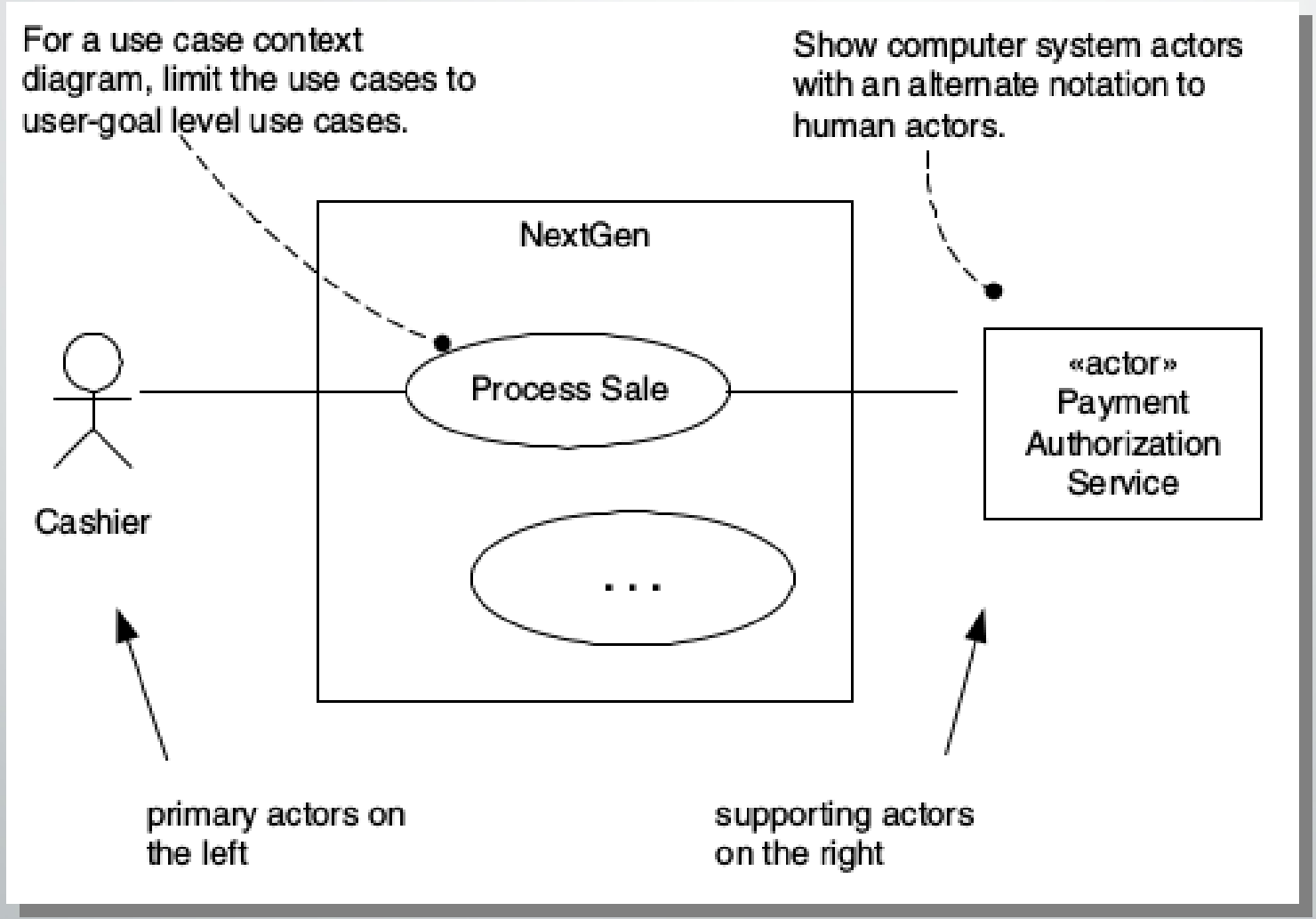
Finding User goals

Actor	Goal	Actor	Goal
Cashier	process sales process rentals handle returns cash in cash out . . .	System Administra-tor	add users modify users delete users manage security manage system tables . . .
Manager	start up shut down . . .	Sales Activity System	analyze sales and per-formance data
. . .	. . .	. . .	. . .

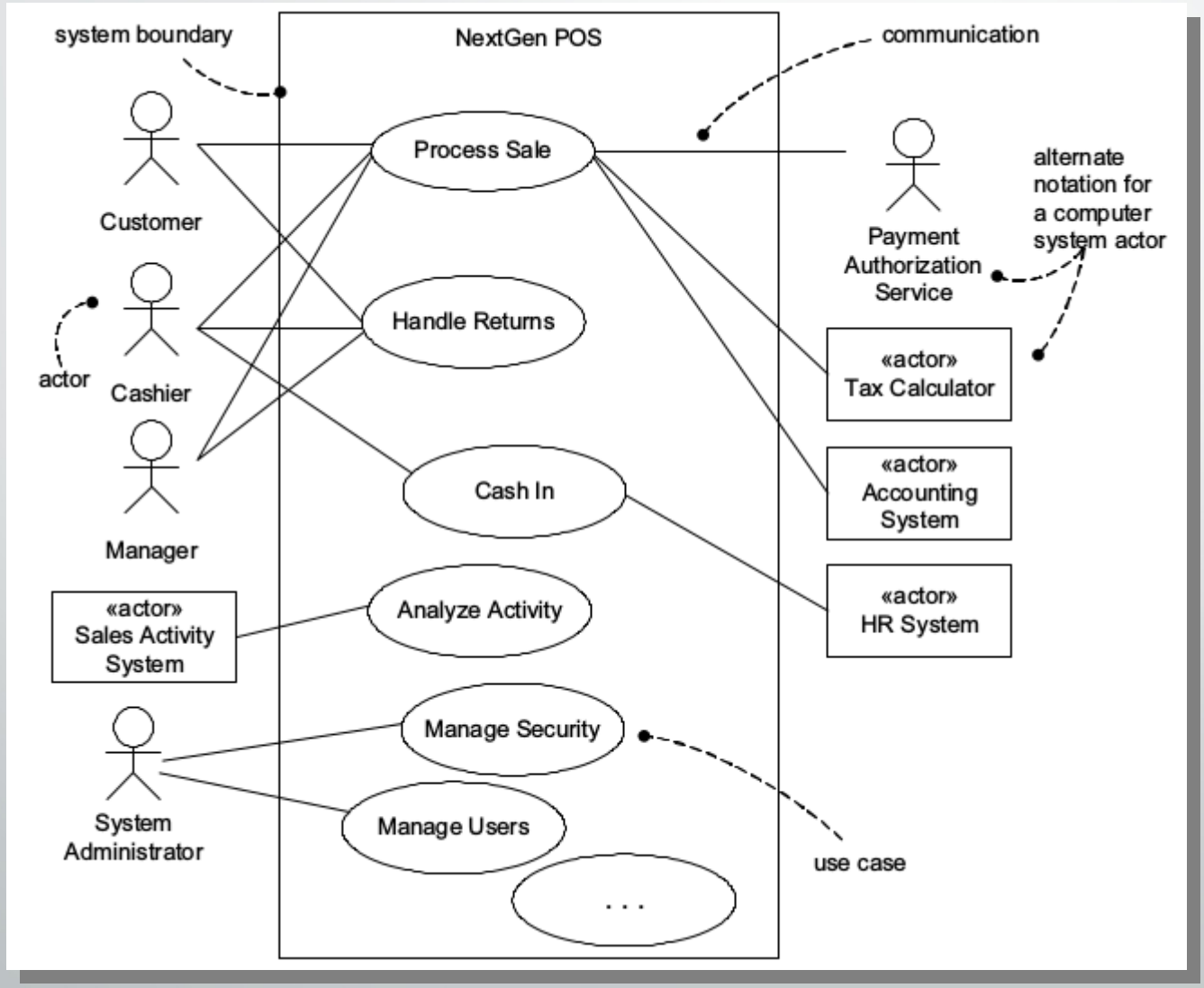
Applying UML: Use Case Diagram

- The UML provides use case diagram notation to illustrate the names of use cases and actors, and the relationships between them.
- Use case diagrams and use case relationships are secondary in use case work.
- Use cases are text documents.
- Doing use case work means to write text.
- Draw a simple use case diagram in conjunction with an actor-goal list.

Applying UML: Use Case Diagram



Applying UML: Use Case Diagram



Generalization, Intension and Extension

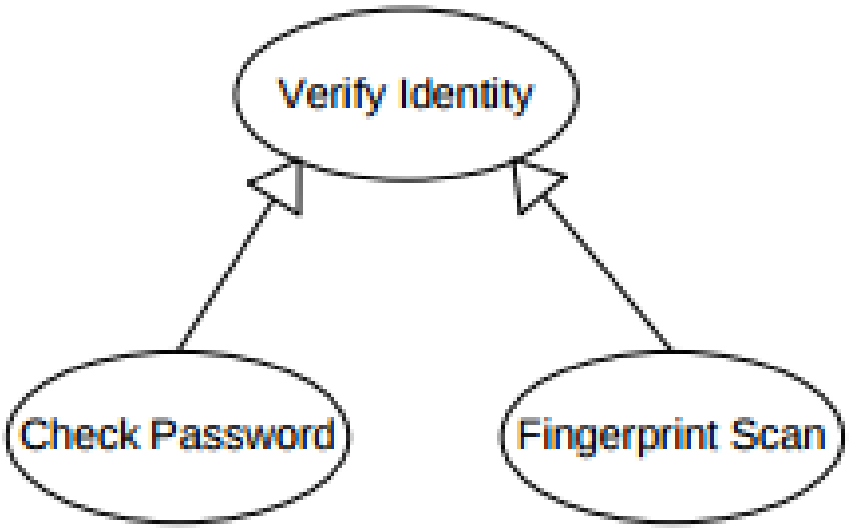
Generalization

A generalized Use Case describes the common of other specialized Use Cases. Used when a number of Use Cases all have some subtasks in common, but each one has something different about it. The generalized and specialized use cases share the same goal.

A specialized Use Case may capture an alternative scenario of the generalized Use Case. The Specialized use case may interact with new actors.

Generalization, Intension and Extension

Generalization example



A Generalization relationship shows an "either/or" condition

Ex. EITHER a Password OR a Fingerprint Scan will verify identity

Generalization, Intension and Extension

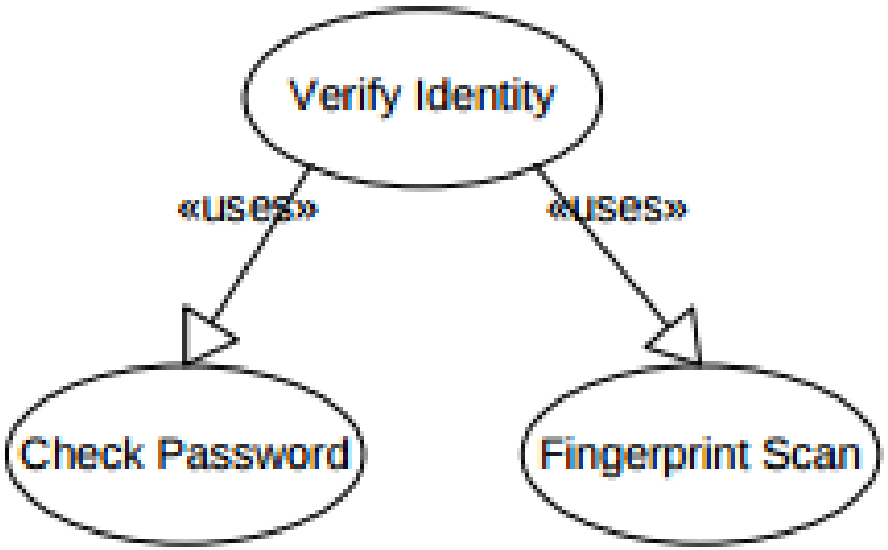
Inclusion

A Use Case is a part of another Use Case. In older versions: “uses”.

When a number of Use Cases have common behavior, which can be modeled in a single use case, inclusion relationship is used. X << includes >> Y indicates that the process of doing X always involves doing Y at least once.

Generalization, Intension and Extension

Inclusion Example:



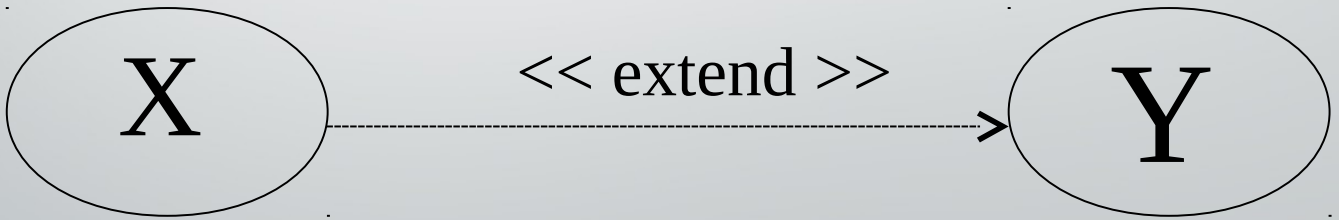
An Includes (or Uses) relationship shows an "AND" condition

Ex. BOTH a Password AND a Fingerprint Scan are required to verify identity

Generalization, Intension and Extension

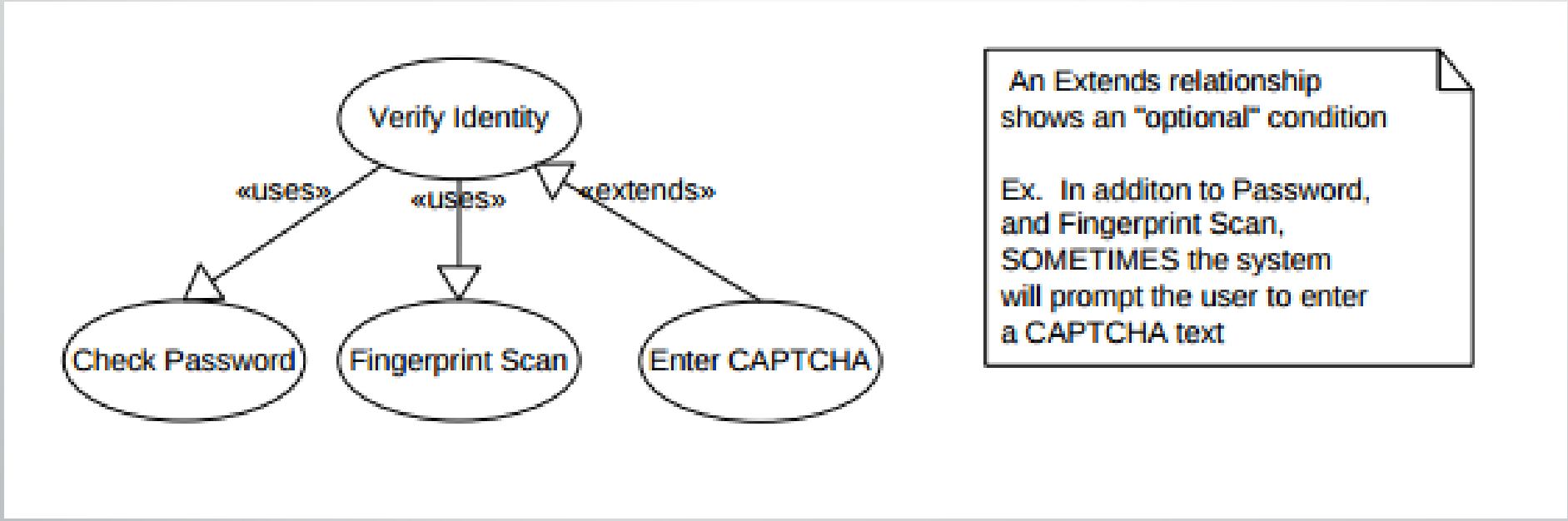
Extension:

A Use Case may extend another Use Case. Serves as extension point to another Use Case. The extended Use Case must explicitly declare its extension points. A different use case handles variations or exceptions from the basic use case. Arrow goes from extended to basic use case.



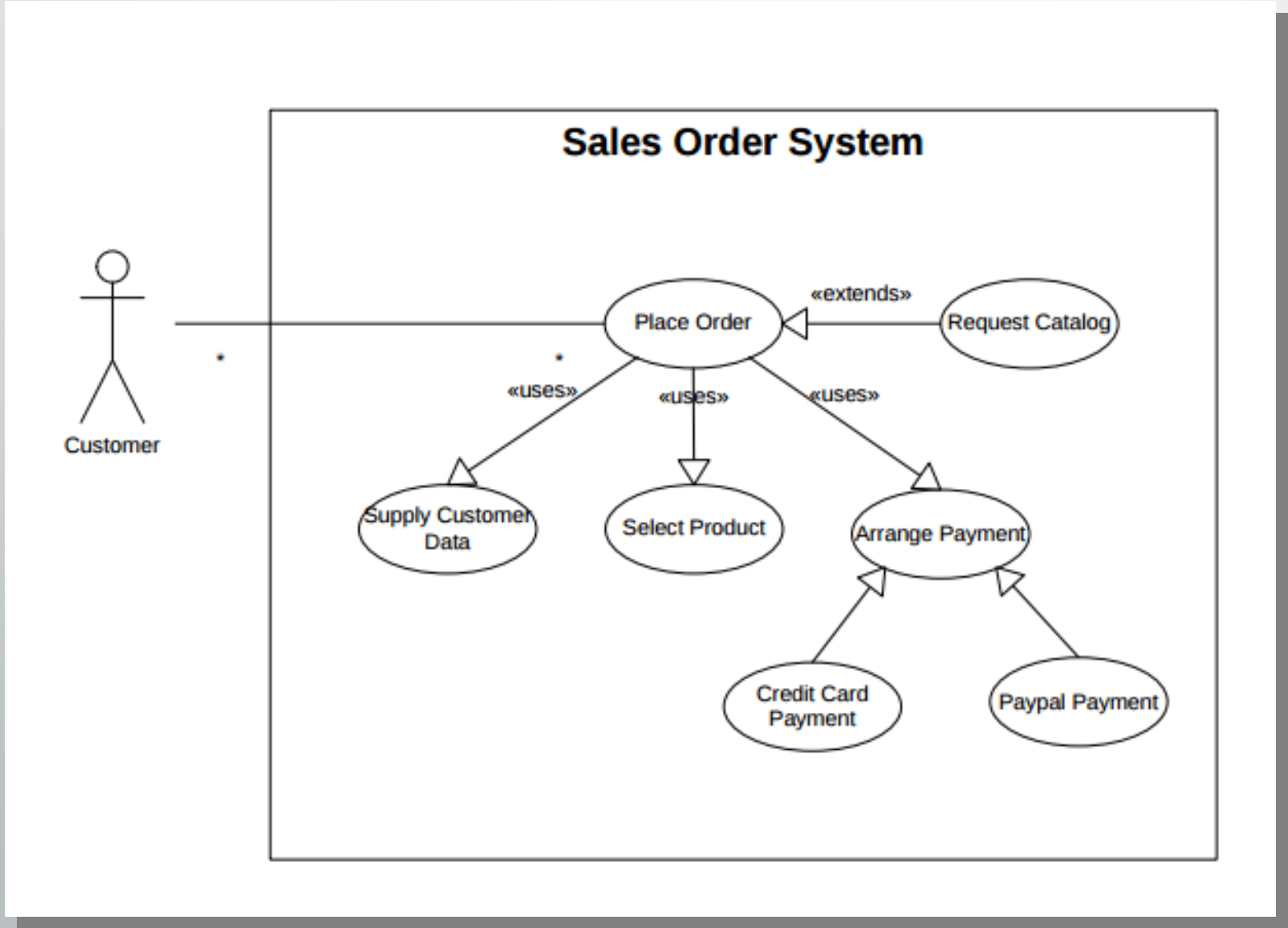
Generalization, Intension and Extension

Extension Example:

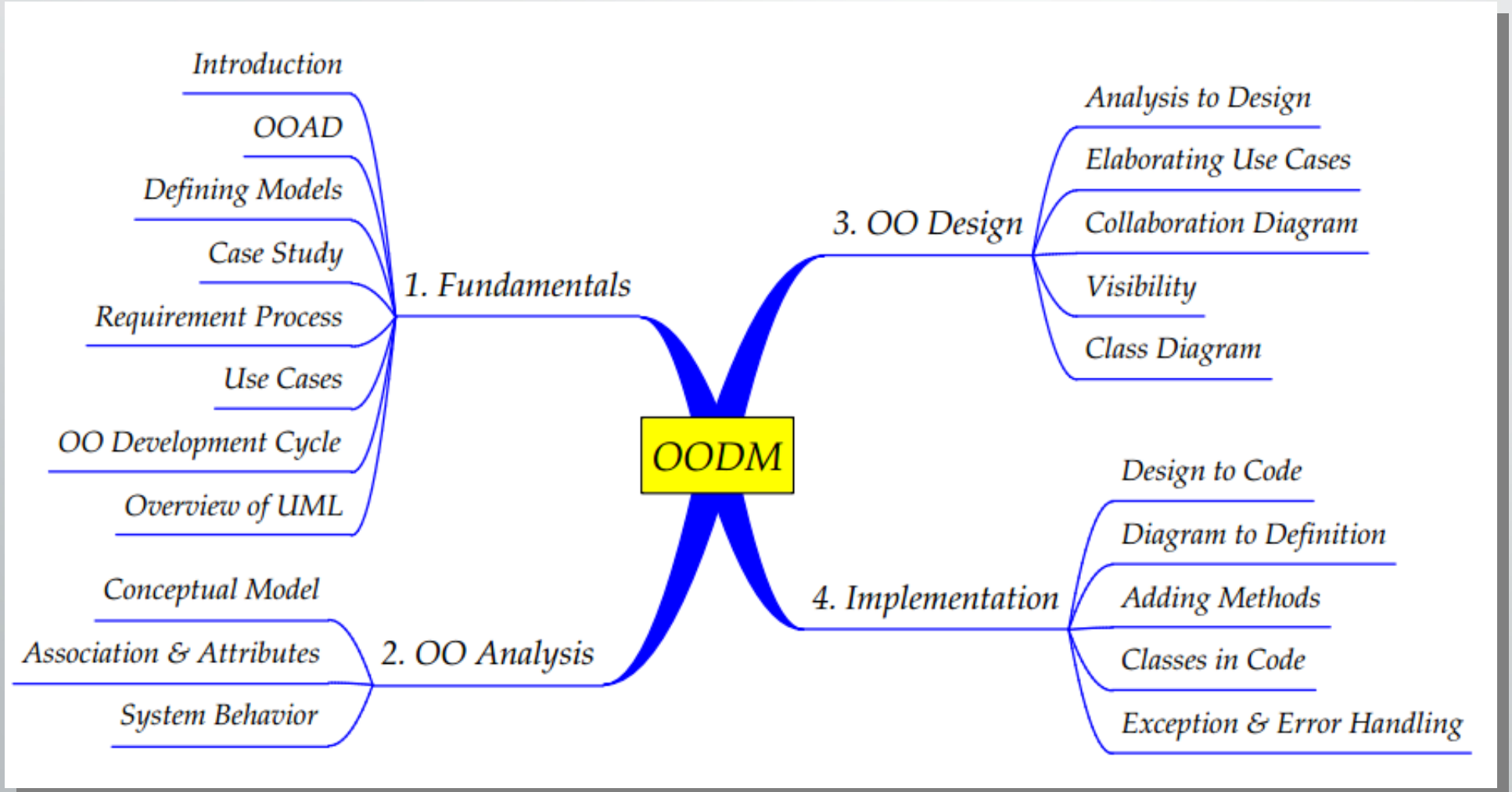


Generalization, Intension and Extension

*An **Use Case Diagram** to illustrate generalization, intension and extension*



Course Content – Abstract



Text Book: **Applying UML and Patterns** by **Craig Larman**

Reference Book: **Software Engineering** by **Ian Sommerville**