CS 123
Introduction to Software Engineering

05: Requirements Engineering

DISCS

SY 2013-2014

Learning Objectives

- To introduce the concept of Requirements Engineering
- To explain the activities involved in the Requirements Gathering Process
- To introduce Requirements Management activities
- To explain how to create Use Cases

Overview

- Requirements Engineering
- Requirements Gathering Process
- Requirements Management
- Use cases

Requirements Engineering

- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called stakeholders.

What is Requirements?

- The requirements themselves are the descriptions of the system's services, constraints and goals by consulting with users/clients.
- Requirements define the function of the system from the client's view point.
- It is defined in a manner that is understandable by both users and development staff

Purpose of requirement

- 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.

Why are Requirements Important?

Causes of failed software projects (Standing Group Study, 1994)

REASON	SHARE
Incomplete Requirements	13.1%
Lack of user involvement	12.4%
Lack of resources	10.6%
Unrealistic expectations	9.9%
Lack of executive support	9.3%
Changing requirements and specifications	8.8%
Lack of planning	8.1%
System no longer needed	7.5%

The most common mistake is to build the wrong system!

Risks of requirements engineering

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

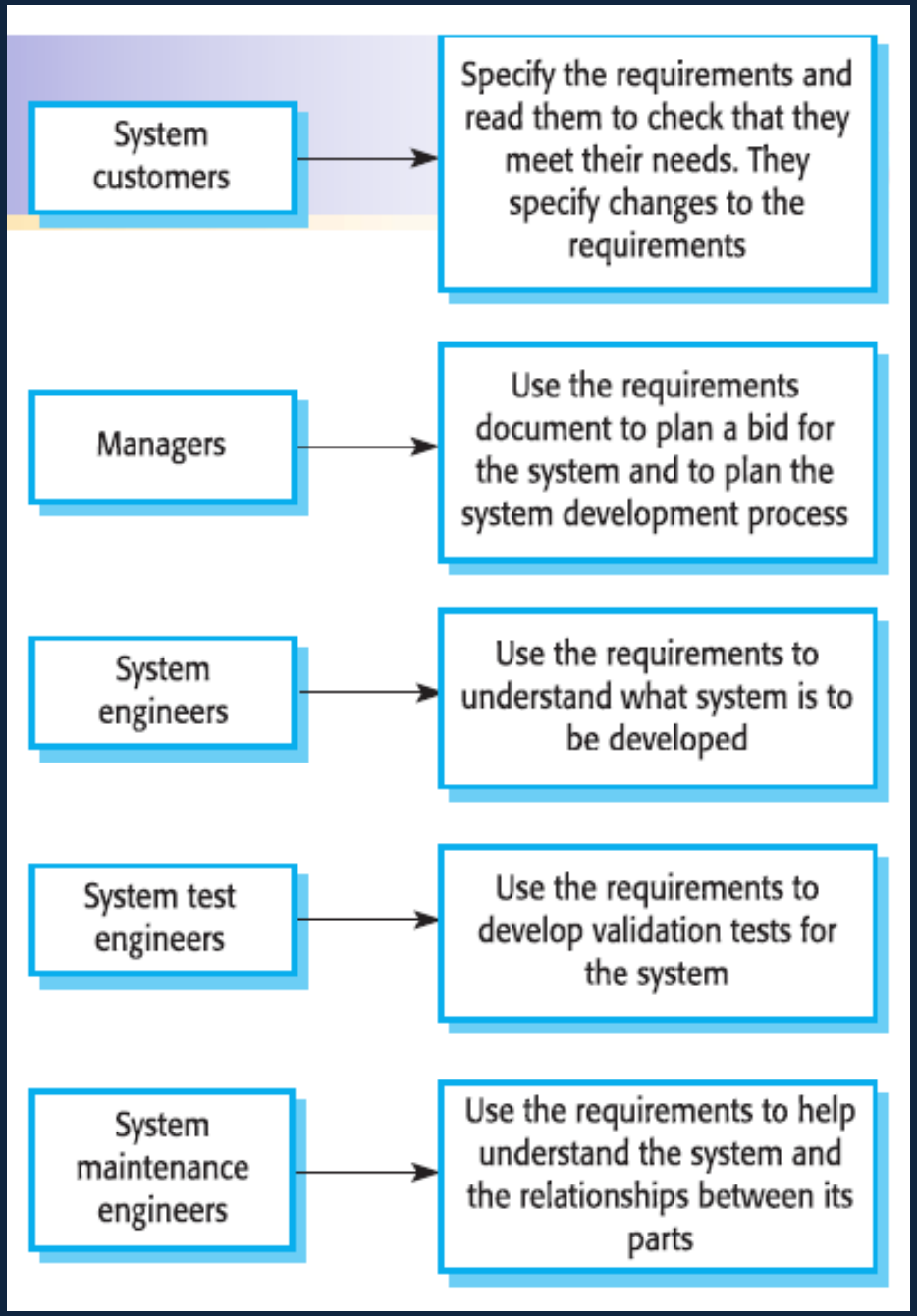
Realism and Verifiability

- Requirements must be **realistic**, i.e., it must be possible to meet them.
  - Not: The system must be capable of x, but no known computer system can do x at a reasonable cost.
- Requirements must be **verifiable**, i.e., it must be possible to test whether a requirement has been met.
  - Wrong: The system must be easy to use.
  - Right: After one day's training an operator should be able to input 50 orders per hour.

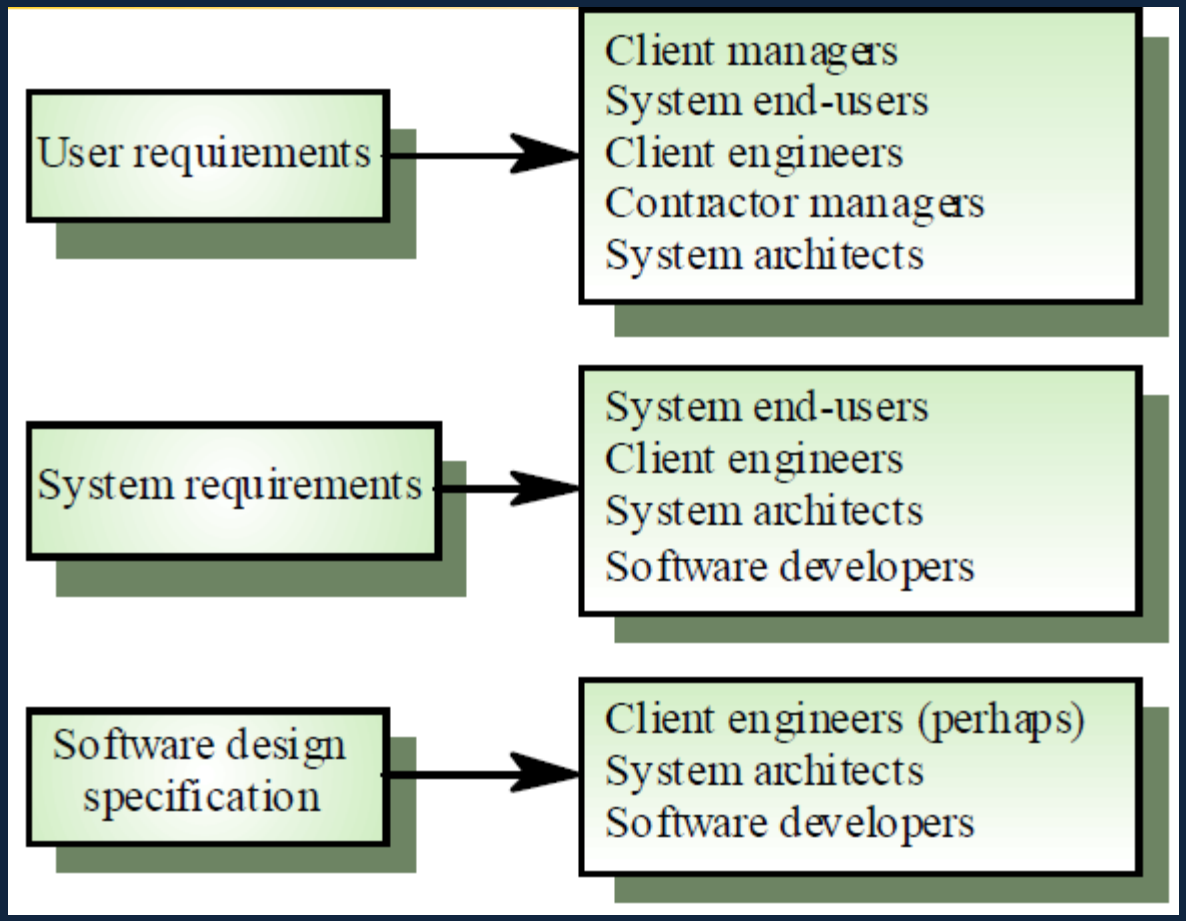
Types of Requirements

- **User requirements**
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints.
  - Written for customers
- **System requirements (specification)**
  - A structured document setting out detailed descriptions of the system services.
  - Written as a contract between client and contractor

Requirements Document Users



Requirements Readers



Initial Requirements

- The initial requirements are based on the initial business model
- Then they are refined
- The requirements are dynamic — there are frequent changes
  - Maintain a list of likely requirements, together with use cases of requirements approved by the client

Initial Requirements

- There are two categories of requirements
  - Functional
  - Non-Functional

Functional Requirements

- A functional requirement specifies an action that the software product must be able to perform
  - Often expressed in terms of inputs and outputs
  - Handled as part of the requirements and analysis workflows
  - Describes system behaviors - what behaviors it does
  - The functions that the system must perform
    - Functionality
    - Data
    - User interfaces

Non-Functional Requirements

- A non-functional requirement specifies properties of the software product itself, such as
  - Platform constraints
  - Response times
  - Reliability
- Some non-functional requirements have to wait until the design workflow
  - The detailed information for some non-functional requirements is not available until the requirements and analysis workflows have been completed

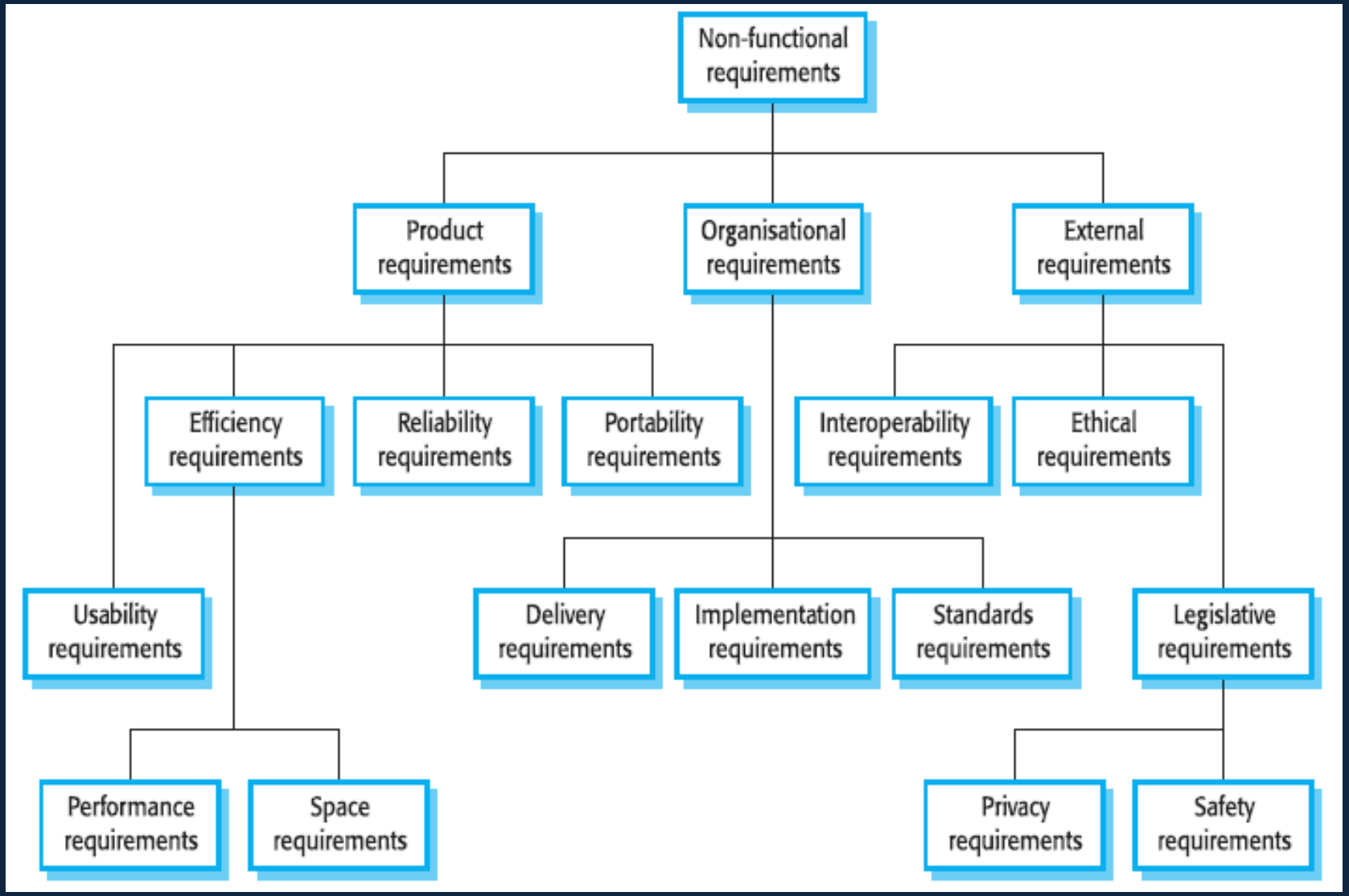
Non-Functional Requirements

- Describes other desired attributes of the overall system - how it does them
- Not directly related to the functions that the system must perform
  - Usability, documentation and training
  - Reliability and quality assurance

Non-functional Requirement Classifications

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational 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.

Non-functional Requirements Types



Non-Functional Requirements Examples

- Product requirement
  - The user interface for LIBSYS shall be implemented as simple HTML without frames or Java applets.
- Organisational requirement
  - The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.
- External requirement
  - The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.

Requirements Documentation

The form of documentation varies, but is likely to contain the following:

- General:
  - Purpose and scope of system
  - Objectives and criteria for success
  - List of terminology, organizations involved, etc.
- Description of current system(s)
- Requirements of proposed system
  - Overview
  - Functional Requirements
  - Non-functional requirements
- System models
  - Scenarios
  - Use cases
  - Models used during analysis

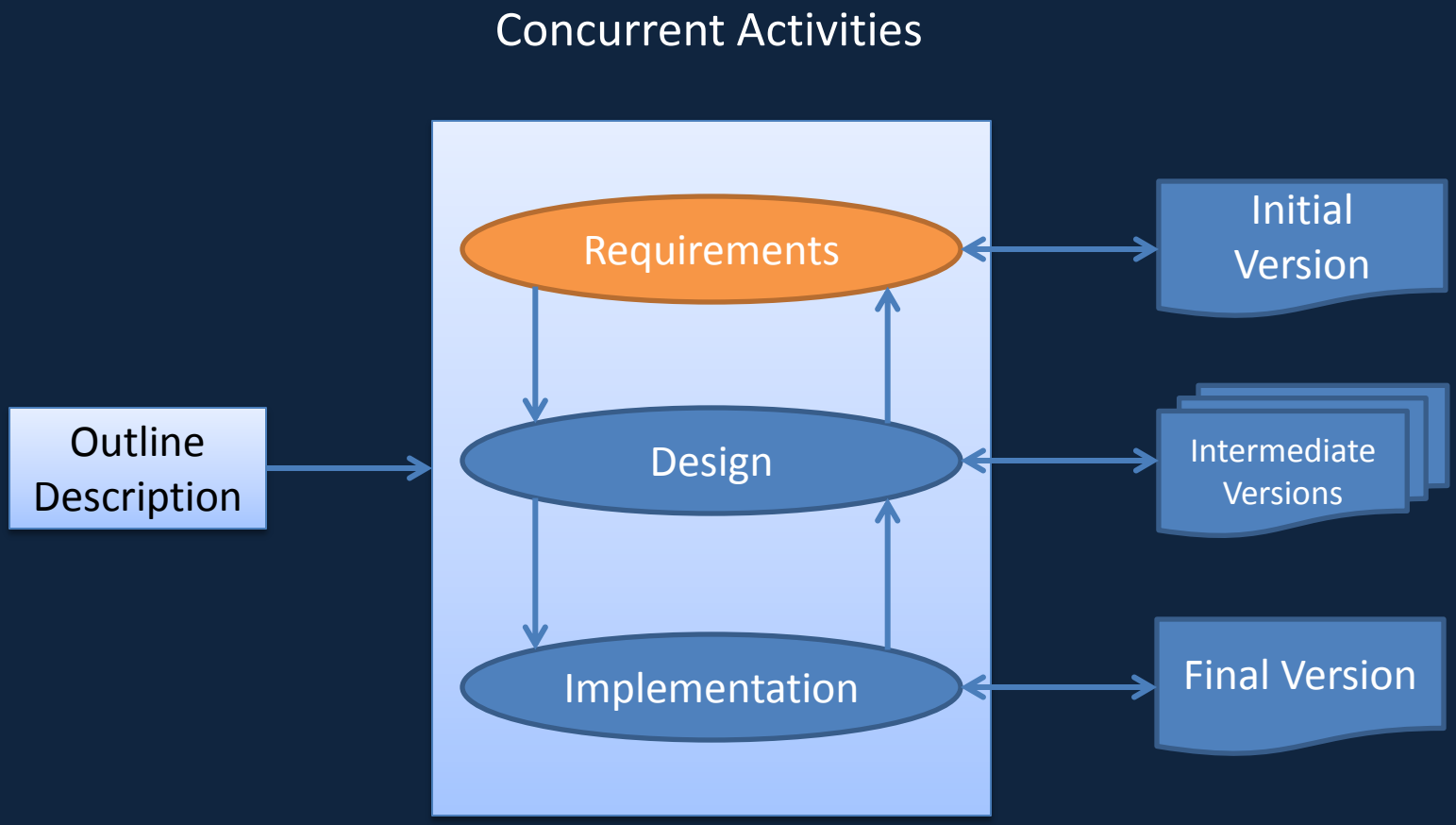
Requirements and Design

- In principle, requirements should state what the system should do and the design should describe how it does this.
- In practice, requirements (incl. Specification) and design are inseparable
  - A system architecture may be designed to structure the requirements;
  - The system may inter-operate with other systems that generate design requirements;
  - The use of a specific design may be a domain requirement.

Overview

- Requirements Engineering
- Requirements Gathering Process
- Requirements Management
- Use cases

Requirements in Iterative Refinement



Evolution of Requirements

If the requirements definition is wrong, the system will be wrong.

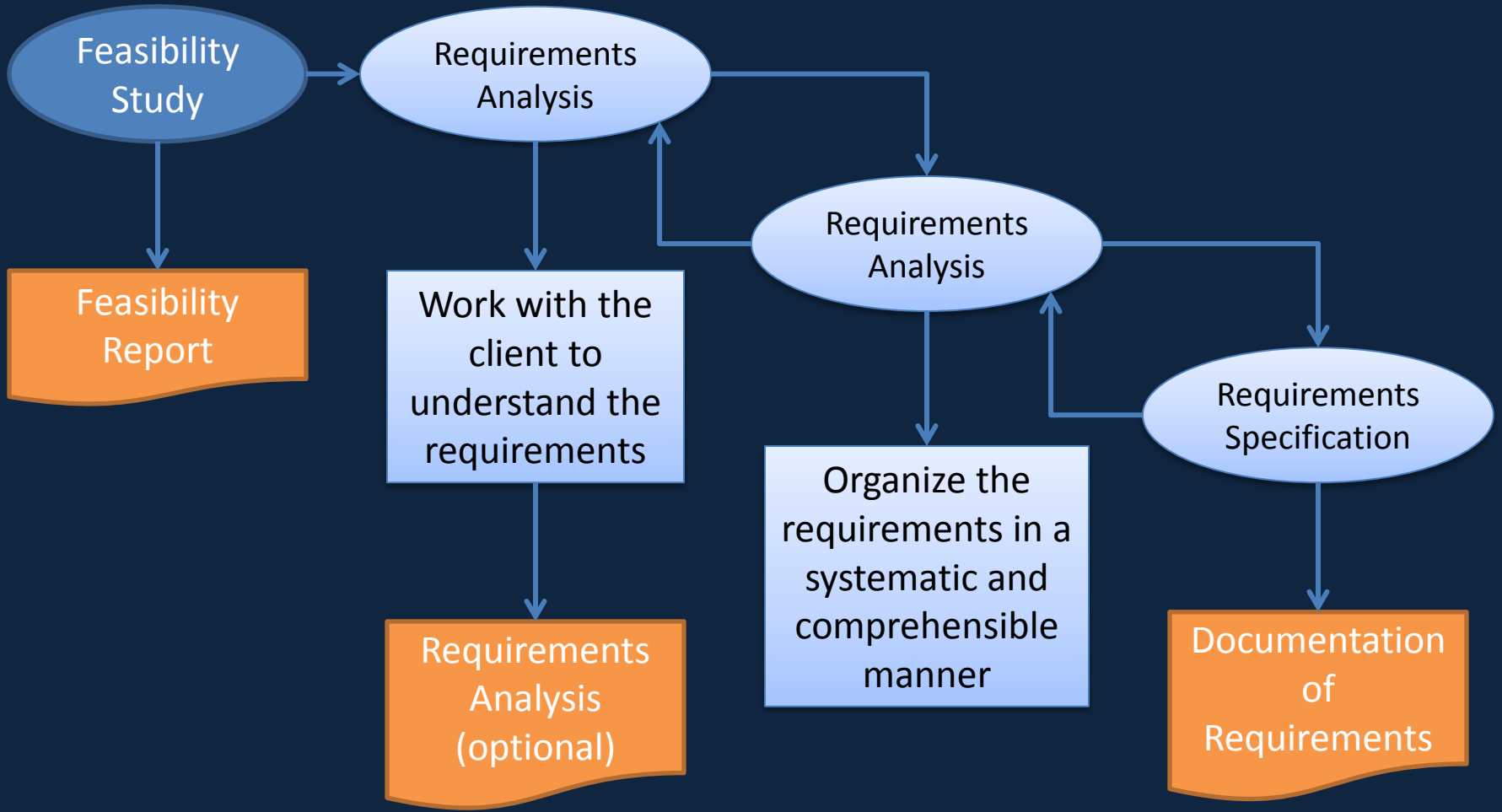
With complex systems, understanding of requirements will always continue to improve.

Therefore…

The requirements definition must evolve.

Its documentation must be kept current (but clearly identify versions).

Requirements Engineering Process



Requirements measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K Bytes Number of RAM 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

Requirements Gathering Process

- Determine what the client needs
- Understand the domain
- Formulate the business model
- Specify the initial requirements

Determine what the client needs

- Misconception
  - We must determine what the client wants
- “I know you believe you understood what you think I said, but I am not sure you realize that what you heard is not what I meant!”
- We must determine what the client needs

Determine what the client needs

- It is hard for a systems analyst to visualize a software product and its functionality
  - The problem is far worse for the client
- A skilled systems analyst is needed to elicit the appropriate information from the client
- The client is the only source of this information

Determine what the client needs

- The solution:
  - Obtain initial information from the client
  - Use this initial information as input to the Requirements Gathering Process
  - Follow the steps of the Requirements Gathering Process to determine the client's real needs

Requirements Workflow

- First, gain an understanding of the application domain (or domain, for short)
  - The specific environment in which the target product is to operate
- Second, build a business model
  - Model the client's business processes
- Third, use the business model to determine the client's requirements
- Iterate the above steps

Definitions

- Discovering the client's requirements
  - Requirements elicitation (or requirements capture)
  - Methods include interviews and surveys
- Refining and extending the initial requirements
  - Requirements analysis

Understanding the Domain

- Every member of the development team must become fully familiar with the application domain
  - Correct terminology is essential
- Construct a glossary
  - A list of technical words used in the domain and their meanings

Business Model

- A business model is a description of the business processes of an organization
- The business model gives an understanding of the client's business as a whole
- This knowledge is essential for advising the client regarding computerization
- The systems analyst needs to obtain a detailed understanding of the various business processes
- Different techniques are used, primarily interviewing

Interviewing

- In formal or informal interviewing, the team puts questions to stakeholders about the system that they use and the system to be developed
- The requirements team meet with the client and users to extract all relevant information

Interviewing

- There are two types of questions
  - Close-ended questions require a specific answer
  - Open-ended questions are posed to encourage the person being interviewed to speak out
- There are two types of interviews
  - In a structured interview, specific pre-planned questions are asked, frequently close-ended
  - In an unstructured interview, questions are posed in response to the answers received, frequently open-ended

Interviewing

- Interviewing is not easy
  - An interview that is too unstructured will not yield much relevant information
  - The interviewer must be fully familiar with the application domain
  - The interviewer must remain open-minded at all times
- After the interview, the interviewer must prepare a written report
  - It is strongly advised to give a copy of the report to the person who was interviewed

Interviews in practice

- Normally a mix of closed and open-ended interviewing.
- Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- Interviews are not good for understanding domain requirements
- Requirements engineers cannot understand specific domain terminology;
- Some domain knowledge is so familiar that people find it hard to articulate or think that it isn’t worth articulating.

Interviewing

- Clients may have only a vague concept of requirements.
  - Prepare before you meet with them
  - Keep full notes
  - If you do not understand, delve further
  - Repeat what you hear
  - Small group meetings are often most effective

Effective interviewers

- Interviewers should be open-minded, willing to listen to stakeholders and should not have pre-conceived ideas about the requirements.
- They should prompt the interviewee with a question or a proposal and should not simply expect them to respond to a question such as “what do you want?”.

Tips

1. ***Identify the stakeholders:***
   - Who is affected by this system?
     - Client
     - Senior management
     - Production staff
     - Computing staff
     - Customers
     - *etc., etc., etc.,*
   - Who can disrupt this project?

Tips

2. ***Understand the requirements in depth:***
   - Domain understanding
   - Understanding of the real requirements of all stakeholders

Tips

3. ***Integrate all view points***

   Example: University Admissions System

   - Applicants
   - University administration
     - Admissions office
     - Financial aid office
     - Special offices (e.g., athletics, development)
   - Computing staff
     - Operations
     - Software development and maintenance
   - Academic departments

Other Techniques

- Interviewing is the primary technique
- A questionnaire is useful when the opinions of hundreds of individuals need to be determined
- Examination of business forms shows how the client currently does business

Other Techniques

- Direct observation of the employees while they perform their duties can be useful
  - Videotape cameras are a modern version of this technique
  - But, it can take a long time to analyze the tapes
  - Employees may view the cameras as an unwarranted invasion of privacy

Overview

- Requirements Engineering
- Requirements Gathering Process
- Requirements Management
- Use cases

Requirements Management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- Requirements are inevitably incomplete and inconsistent
  - New requirements emerge during the process as business needs change and a better understanding of the system is developed;
  - Different viewpoints have different requirements and these are often contradictory.

Requirements Change Management

- Should apply to all proposed changes to the requirements.
- Principal stages
  - Problem analysis. Discuss requirements problem and propose change;
  - Change analysis and costing. Assess effects of change on other requirements;
  - Change implementation. Modify requirements document and other documents to reflect change.

Change management

Requirements reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

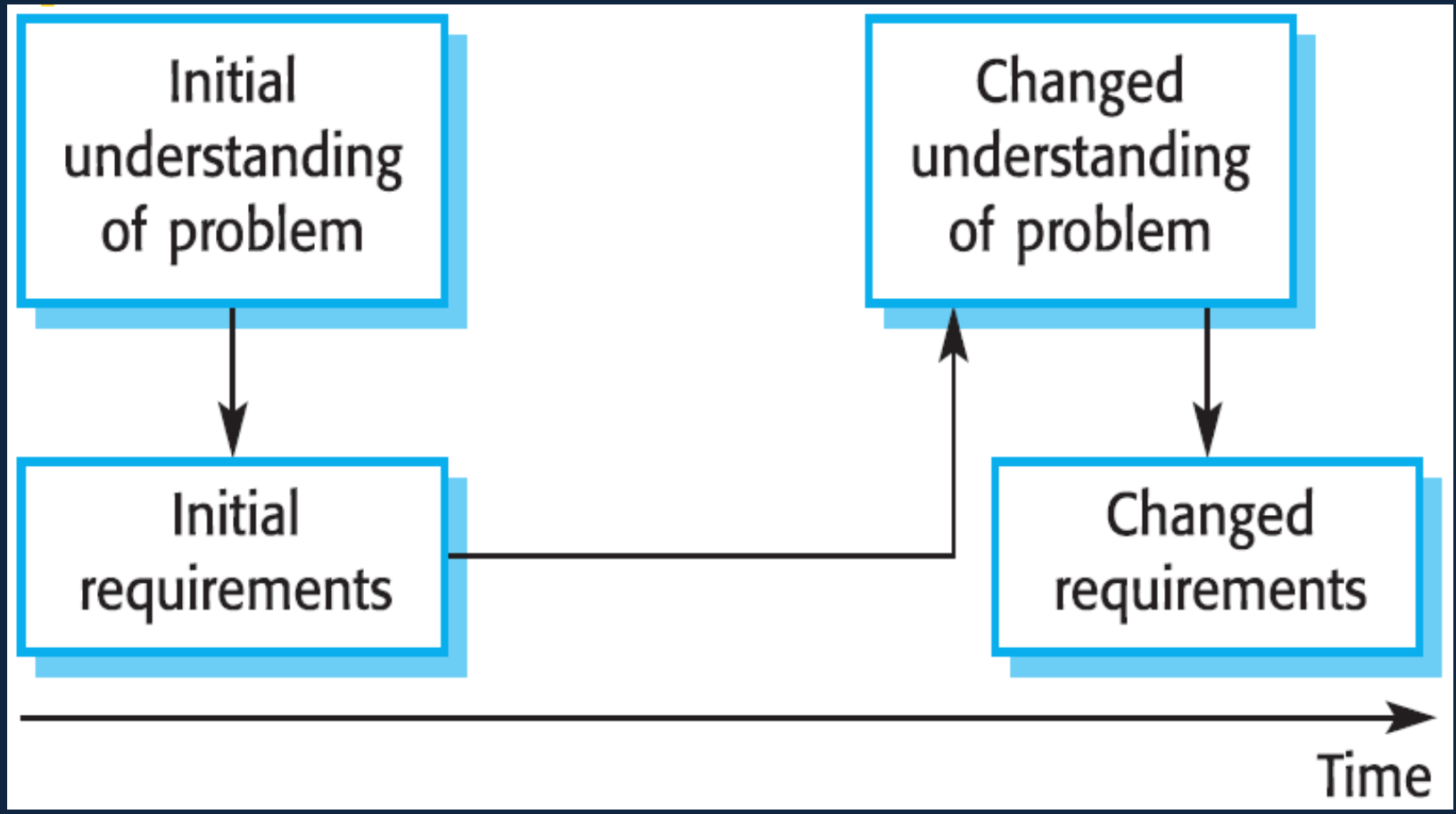
Review checks

- Verifiability. Is the requirement realistically testable?
- Comprehensibility. Is the requirement properly understood?
- Traceability. Is the origin of the requirement clearly stated?
- Adaptability. Can the requirement be changed without a large impact on other requirements?

Requirements change

- The priority of requirements from different viewpoints changes during the development process.
- System customers may specify requirements from a business perspective that conflict with end-user requirements.
- The business and technical environment of the system changes during its development.

Requirements evolution



Enduring and volatile requirements

- Enduring requirements. Stable requirements derived from the core activity of the customer organisation. E.g. a hospital will always have doctors, nurses, etc. May be derived from domain models
- Volatile requirements. Requirements which change during development or when the system is in use. In a hospital, requirements derived from health-care policy

Requirements Management Planning

- During the requirements engineering process, you have to plan for:
  - Requirements identification
    - How requirements are individually identified;
  - A change management process
    - The process followed when analysing a requirements change;
  - Traceability policies
    - The amount of information about requirements relationships that is maintained;
  - CASE tool support
    - The tool support required to help manage requirements change;

Overview

- Requirements Engineering
- Requirements Gathering Process
- Requirements Management
- Use cases

User-Centred Design

- Software development should focus on the needs of the users.
  - Understand your users
  - Design software based on an understanding of the users' tasks
  - Ensure users are involved in decision making processes
  - Design the user interface following the guidelines for good usability
  - Have users work with and give their feedback about prototypes, on-line help and draft user manuals.

Using Use Case Diagrams

- 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.
- Use case diagrams contain use cases, actors, and their relationships.

Use Case

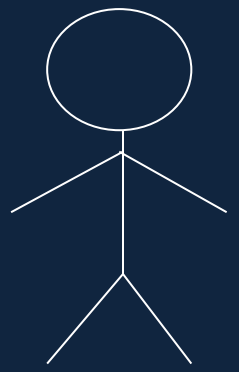
- Use cases specify desired behavior.
- A use case is a description of a set of sequences of actions, including variants, a system performs to yield an observable result of value to an actor.
- Each sequence represent an interaction of actors with the system.

Specifying the Behavior of a Use Case

- Describing the flow of events within the use case.
- Can be done in natural language, formal language or pseudo-code.
- Includes: how and when the use case starts and ends; when the use case interacts with actors and what objects are exchanged; the basic flow and alternative flows of the behavior.

Actors

- An actor represents a set of roles that users of use case play when interacting with these use cases.
- Actors can be human or automated systems.
- Actors are entities which require help from the system to perform their task or are needed to execute the system's functions.
- Actors are not part of the system.

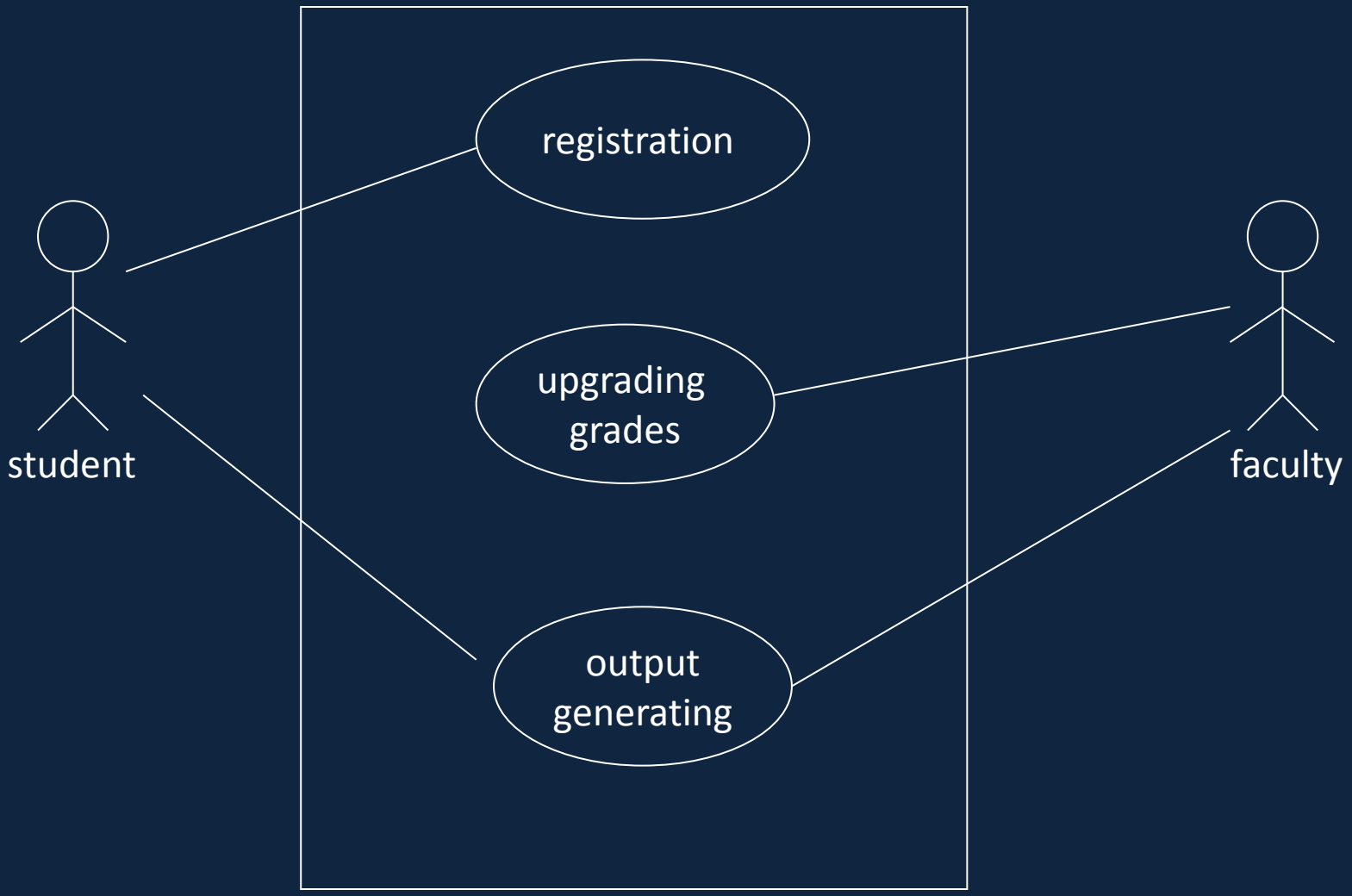


Actor Name

Use Cases and Actors

- From the perspective of a given actor, a use case does something that is of value to the actor, such as calculate a result or change the state of an object.
- The Actors define the environments in which the system lives

Example of Use Case Diagram

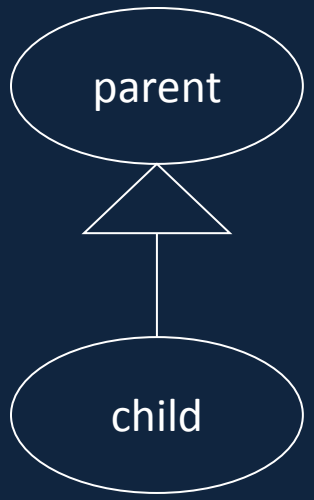


Relationships between Use Cases

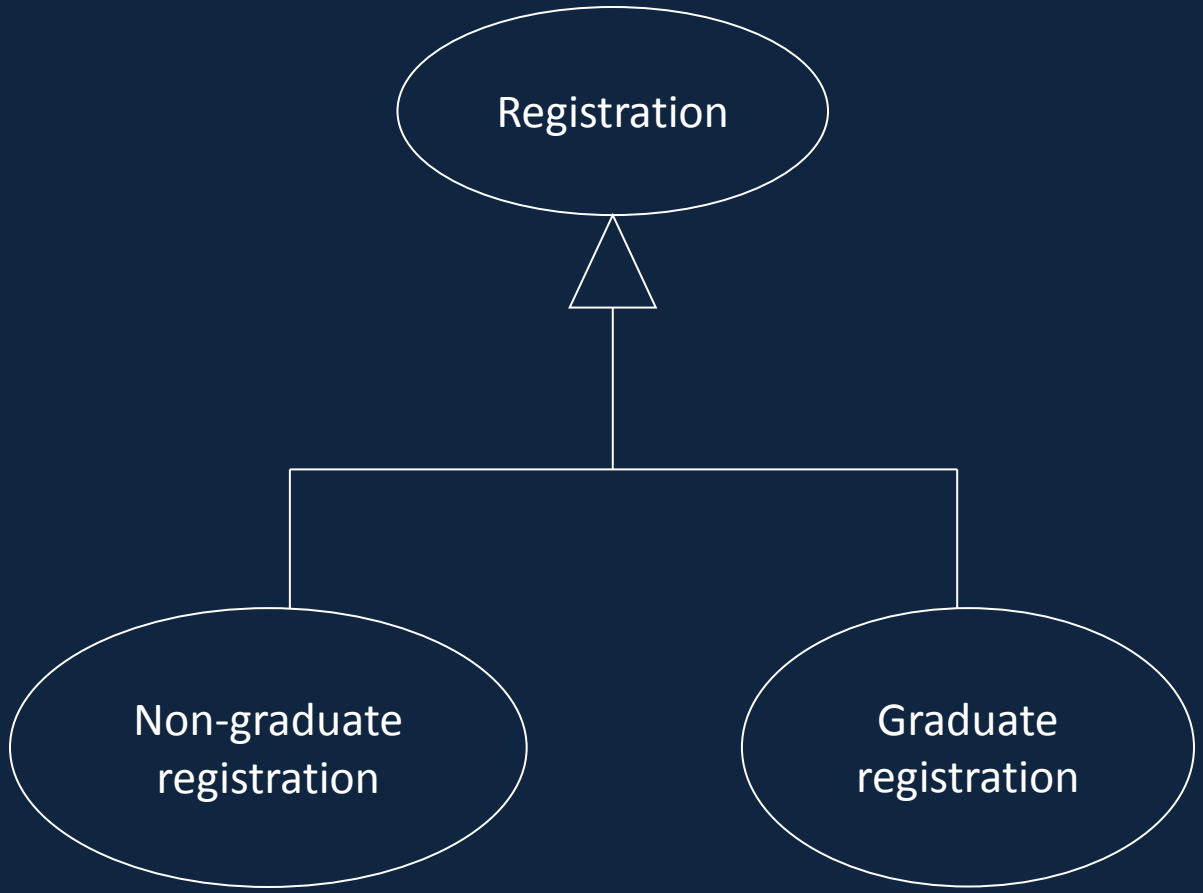
1. Generalization - use cases that are specialized versions of other use cases.
2. Include - use cases that are included as parts of other use cases. Enable to factor common behavior.
3. Extend - use cases that extend the behavior of other core use cases. Enable to factor variants.

1. Generalization

- The child use case inherits the behaviour and meaning of the parent use case.
- The child may add to or override the behaviour of its parent.



More about Generalization

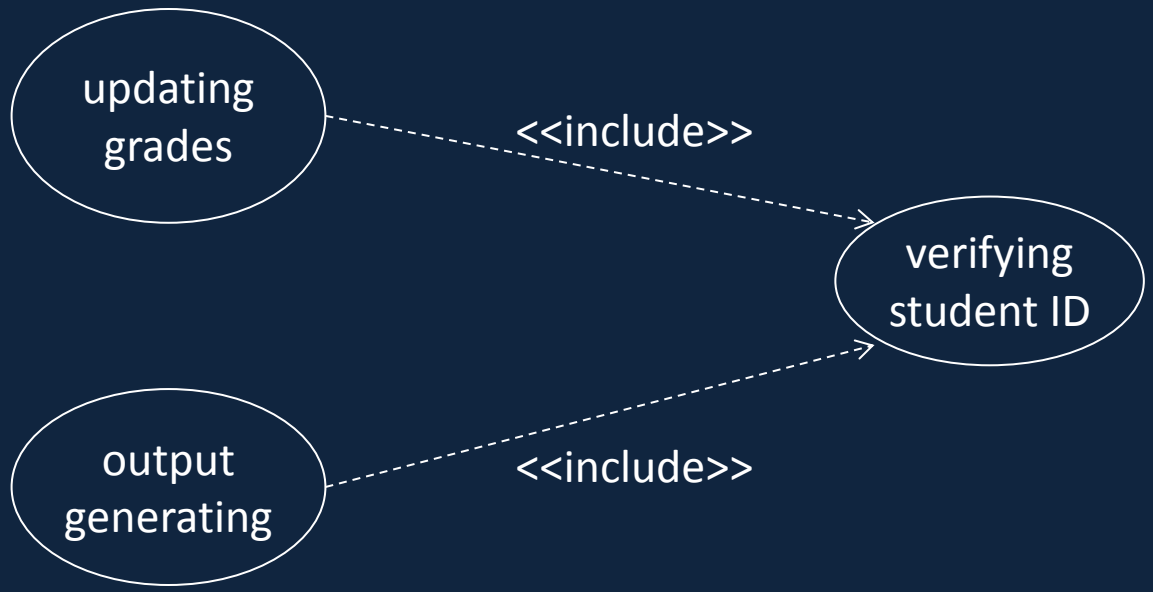


2. Include

- The base use case explicitly incorporates the behaviour of another use case at a location specified in the base.
- The included use case never stands alone. It only occurs as a part of some larger base that includes it.

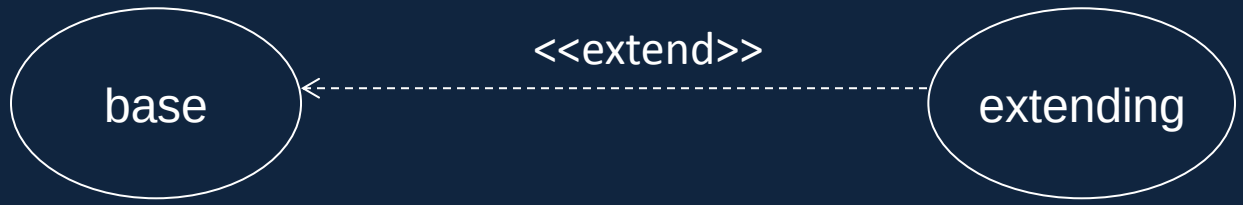
More about Include

- Enables to avoid describing the same flow of events several times by putting the common behaviour in a use case of its own.



3. Extend

- The base use case implicitly incorporates the behaviour of another use case at certain points called extension points.
- The base use case may stand alone, but under certain conditions its behaviour may be extended by the behaviour of another use case.

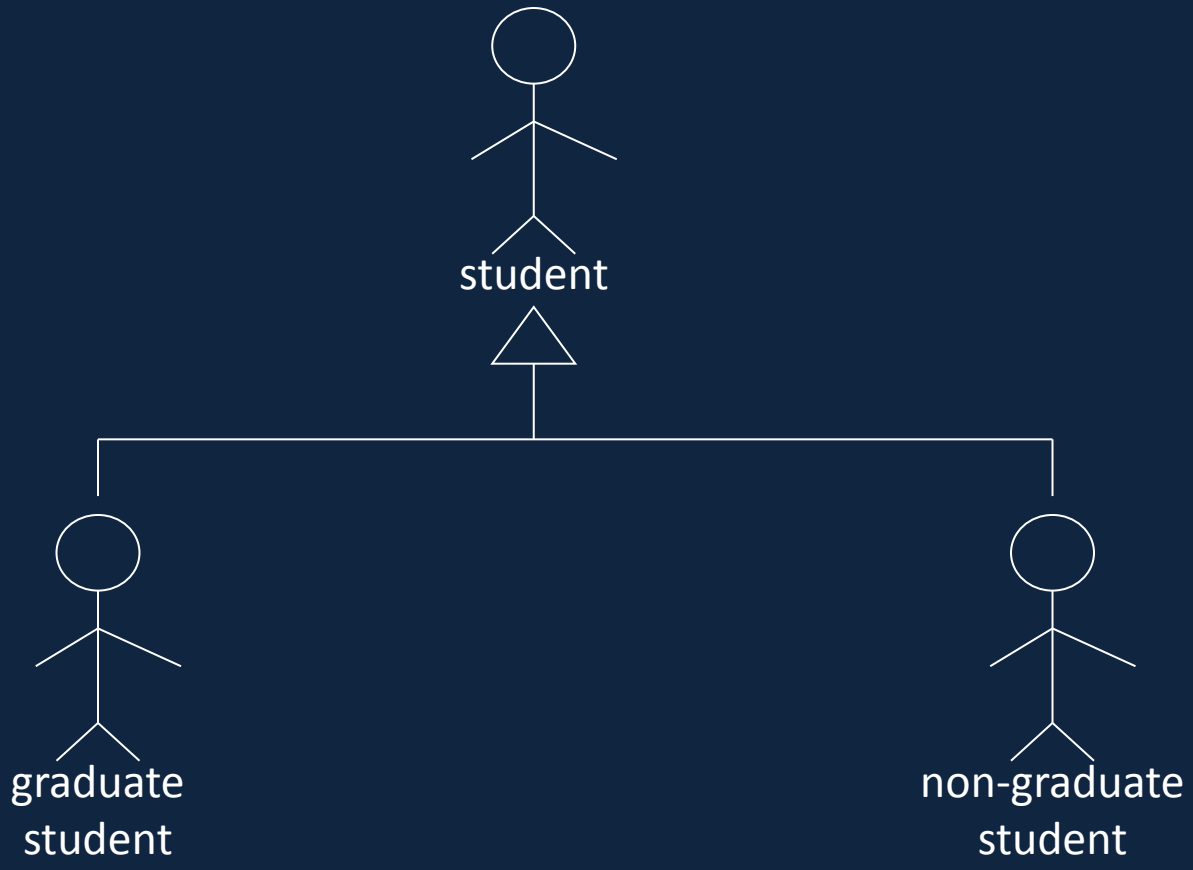


More about Extend

- Enables to model optional behaviour or branching under conditions.

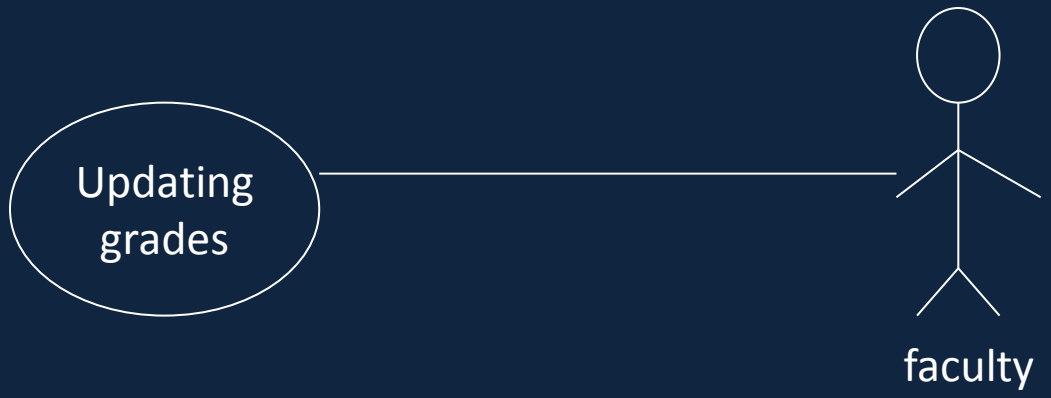
Relationships between Actors

- Generalization.

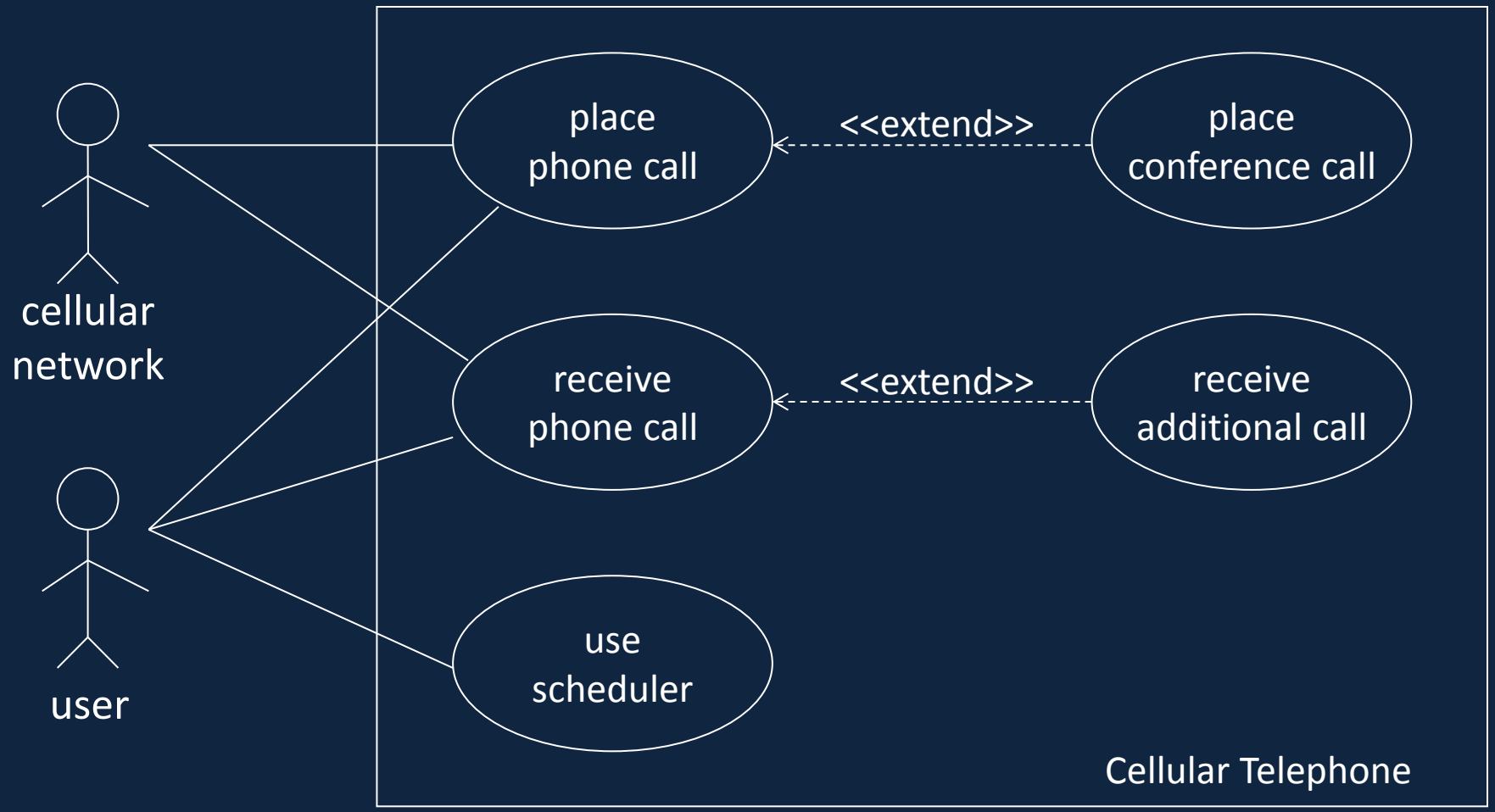


Relationships between Use Cases and Actors

- Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.



Example


cellular network
user
place phone call
<<extend>>
place conference call
receive phone call
<<extend>>
receive additional call
use scheduler
Cellular Telephone

Use Case Description

- Each use case may include all or part of the following:
  - Title or Reference Name – meaningful name of the UC
  - Author/Date – the author and creation date
  - Modification/Date – last modification and its date
  - Purpose – specifies the goal to be achieved
  - Overview – short description of the processes
  - Cross References – requirements references
  - Actors – agents participating
  - Pre Conditions – must be true to allow execution
  - Post Conditions – will be set when completes normally
  - Normal flow of events – regular flow of activities
  - Alternative flow of events – other flow of activities
  - Exceptional flow of events – unusual situations
  - Implementation issues – foreseen implementation problems

Example- Money Withdraw

- 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- Money Withdraw

- 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- Money Withdraw

- 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- Money Withdraw

- 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

Example- Money Withdraw

- One method to identify use cases is actor-based:
  - Identify the actors related to a system or organization.
  - For each actor, identify the processes they initiate or participate in.
- A second method to identify use cases is event-based:
  - Identify the external events that a system must respond to.
  - Relate the events to actors and use cases.
- The following questions may be used to help identify the use cases for a system:
  - What are tasks of each actor ?
  - Will any actor create, store, change, remove, or read information in the system?
  - What use cases will create, store, change, remove, or read this information ?
  - Will any actor need to inform the system about sudden, external changes ?
  - Does any actor need to be informed about certain occurrences in the system ?
  - Can all functional requirements be performed by the use cases ?

Moving on

- The “things” that “live” inside the system are responsible for carrying out the behavior the actors on the outside expect the system to provide.
- To implement a use case, we create a society of classes that work together to carry out the behaviour of the use case.

References

- CS 123 Lectures: Kardi Teknomo, PhD
- Schach. Object-Oriented and Classical Software Engineering, 8th Ed.