4. Requirements Engineering

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

Requirements Engineering

- The requirements for a system are the descriptions of the services that a system should provide and the constraints on its operation.
- These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order, or finding information.
- The process of finding out, analyzing, documenting and checking these services and constraints is called requirements engineering (RE).

What is a Requirement?

- The term requirement is not used consistently in the software industry.
- In some cases, a requirement is simply a high-level, abstract statement of a service that a system should provide or a constraint on a system.
- At the other extreme, it is a detailed, formal definition of a system function.

Requirements Abstraction (Davis, Researcher)

"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."

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**
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

User and System Requirements

User requirements definition

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

System requirements specification

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

Project Stakeholders

- A project stakeholder is any individual or an organization that is actively involved in a project, or whose interest might be affected (positively or negatively) as a result of project execution or completion.

Types of Stakeholders

- The project manager
- The project team
- The project sponsor
- The performing organizations
- The partners
- The client
- The “rest”: anyone who might be affected by the project outputs

System Stakeholders for the Mentcare System

- Patients whose information is recorded in the system and relatives of these patients.
- 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.

4.1 Functional & Non-Functional Requirements

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

Functional & Non-Functional Requirements

- **Functional requirements** (what the system should do)
  - 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** (how the system performs a certain function)
  - 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.
- These requirements depend on the type of software, expected users of the software.
- Functional user requirements may be high-level statements of what the system should do.
- Functional requirements should be written in natural language so that system users and managers can understand them.
- Functional system requirements expand the user requirements and are written for system developers.

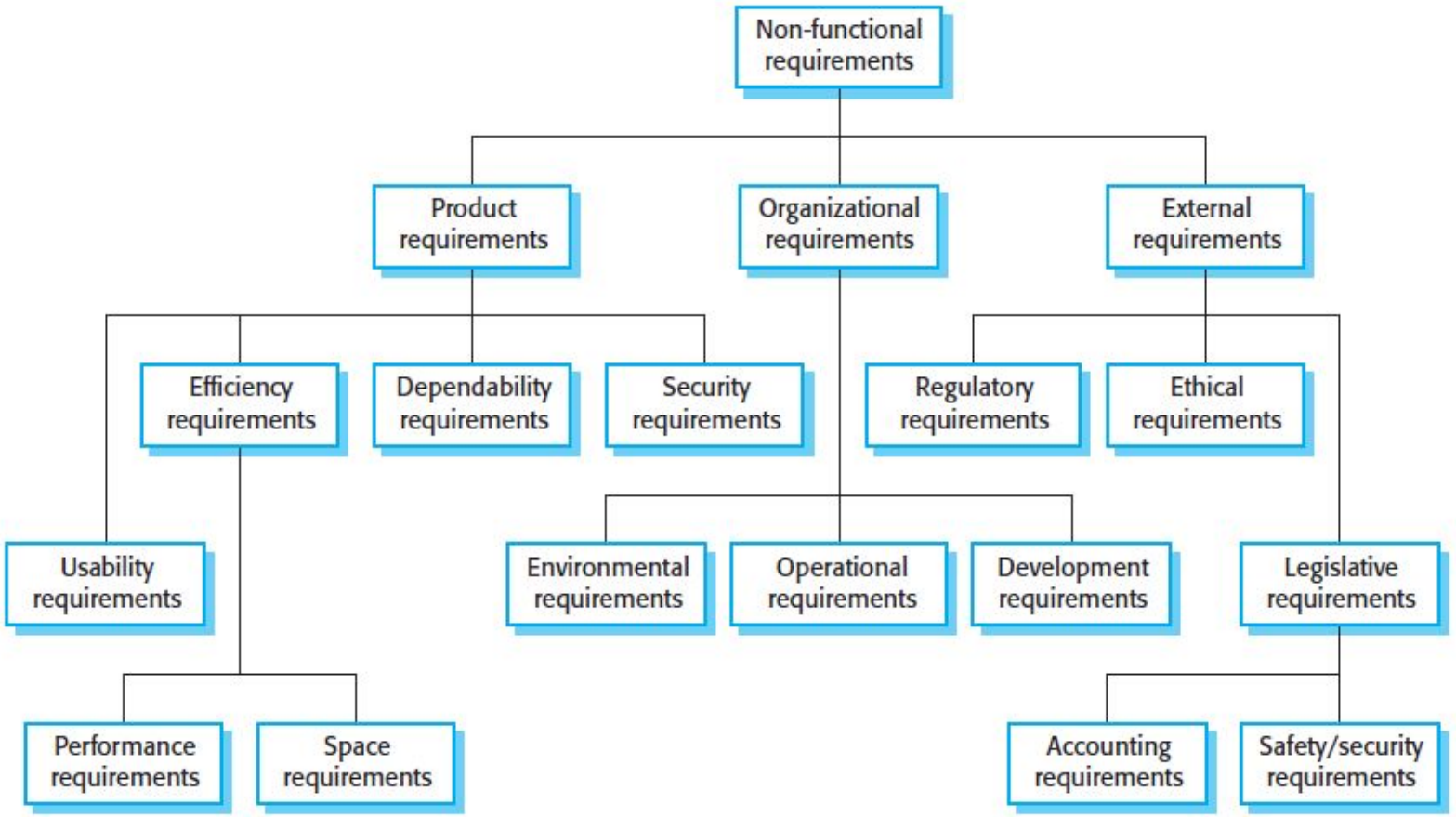
Example: Mentcare System

- Used to maintain information about patients receiving treatment for mental health problems.
- 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.

Non-Functional Requirements

- Non-functional requirements are requirements that are not directly concerned with the specific services delivered by the system to its users.
- Non-functional requirements specify the system's 'quality characteristics' or 'quality attributes'.
- NFR usually specify or constrain characteristics of the system as a whole.
- They may relate to emergent system properties such as reliability, response time, and memory use.

Types of Non-Functional Requirements



Implementation of Non-Functional Requirements

- Non-functional requirements may affect the overall architecture of a system rather than the individual components.
  - For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.

Classification of Non-Functional Requirements

1. **Product requirements**
   - These requirements specify or constrain the runtime behavior of the software.
   - Examples include performance requirements for;
     - how fast the system must execute
     - how much memory it requires
   - reliability requirements that set out the acceptable failure rate
   - security requirements
   - usability requirements
2. **Organizational requirements**
   - These requirements are broad system requirements derived from policies and procedures in the customer's and developer's organizations.
   - Examples include operational process requirements that define how the system will be used
   - development process requirements that specify the programming language;

Classification of Non-Functional Requirements

3. External requirements

- This broad heading covers all requirements that are derived from factors external to the system and its development process.
- These may include regulatory requirements that set out what must be done for the system to be approved for use by a regulator, such as a nuclear safety authority;
- legislative requirements that must be followed to ensure that the system operates within the law; and ethical requirements that ensure that the system will be acceptable to its users and the general public

Example: Mentcare System

- Product requirement
  - The Mentcare system shall be available to all clinics during normal working hours (Mon–Fri, 08:30–17:30).
  - Downtime within normal working hours shall not exceed 5 seconds in any one day.
- Organizational requirement
  - Users of the Mentcare system shall identify themselves using their health authority identity card.
- External requirement
  - The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

List of Non-Functional Requirements

- Performance
  Example: Response Time, Throughput, Utilization, Static Volumetric
- Scalability
- Capacity
- Availability
  - uptime vs downtime
- Reliability
  - software fails under certain conditions
- Recoverability
- Maintainability
- confidentiality

List of Non-Functional Requirements

- Safety
- Environmental
- Interoperability
  - ability to exchange information and communicate with internal and external applications and systems.
- Accuracy and Precision (Data Integrity)
  - Requirements about the accuracy and precision of the data.
- Portability
  - The effort required to move the software to a different target platform. The measurement is most commonly person-months or % of modules that need changing.

List of Non-Functional Requirements

- Security
  - One or more requirements about protection of your system and its data.
- Usability
  - Requirements about how difficult it will be to learn and operate the system. The requirements are often expressed in learning time or similar metrics.
- Legal
  - There may be legal issues involving privacy of information, intellectual property rights, export of restricted technologies, etc.

Activity

Consider the features of an Airline Reservation System and

- Write down all the Stakeholders in this system
- Identify main sub systems of the proposed system
- Write down functional requirements and explain
- Write down non-functional requirements and explain

4.2 Requirements Engineering Processes

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

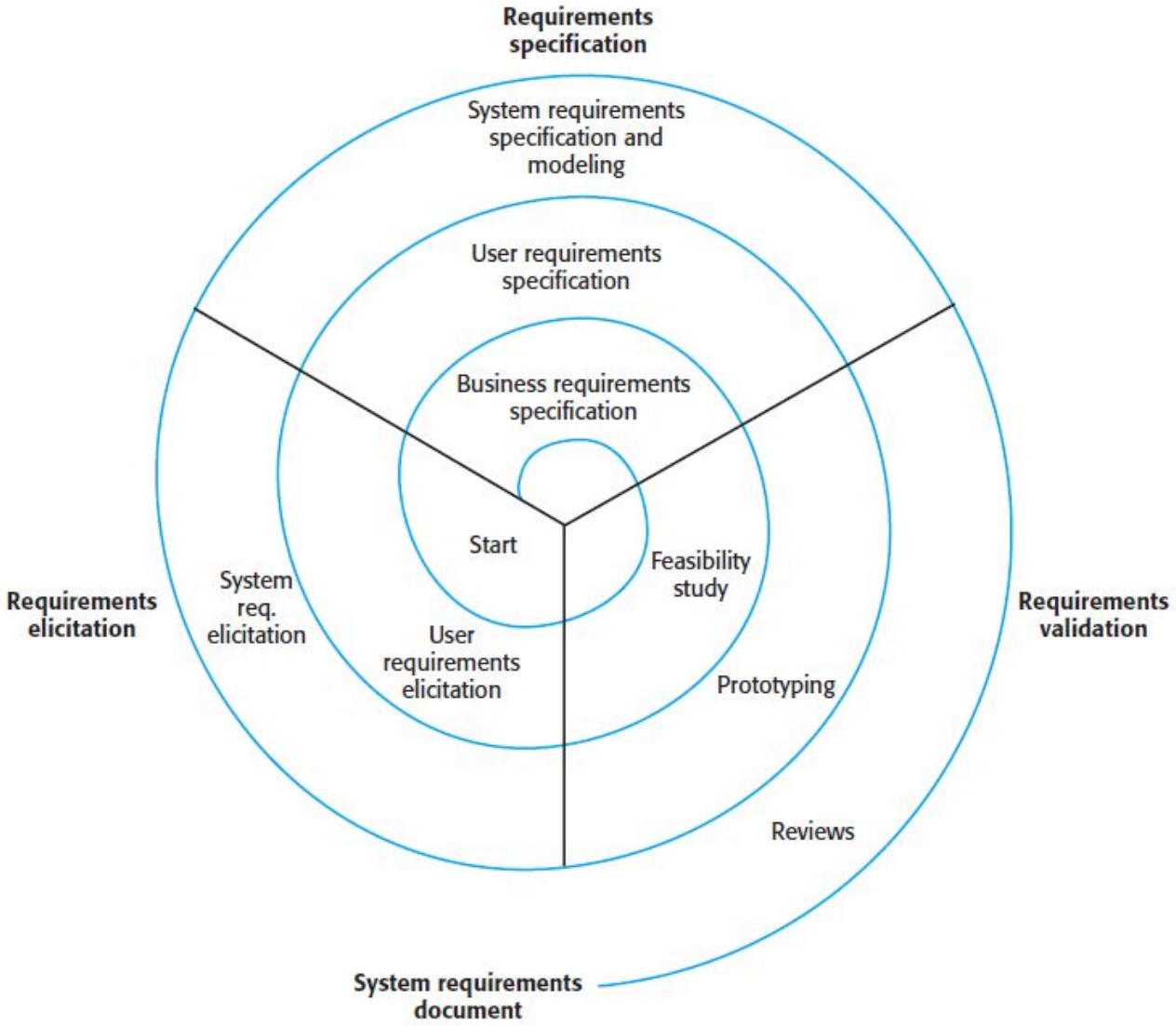
Requirements Engineering Processes

- Requirements engineering involves three key activities

1. **Requirements elicitation and analysis**
   - Discover requirements by interacting with stakeholders
2. **Requirements Specification**
   - converting these requirements into a standard form
3. **Requirements Validation**
   - checking that the requirements actually define the system that the customer wants

- In practice, RE is an iterative process in which these processes are interleaved.

A Spiral View of the Requirements Engineering Process



4.3 Requirements Elicitation

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

Requirements Elicitation and Analysis

- The aims of the requirements elicitation process are to understand the work that stakeholders do and how they might use a new system to help support that work.

- During requirements elicitation, software engineers work with stakeholders to find out about the application domain, work activities, the services and system features that stakeholders want, the required performance of the system, hardware constraints, and so on.

Problems of Requirements Analysis

- 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.

Requirements Elicitation Techniques

- There are two fundamental approaches to requirements elicitation:
  1. Interviewing
     - where you talk to people about what they do.
  2. Observation or ethnography
     - where you watch people doing their job to see what artifacts they use, how they use them, and so on.
- You should use a mix of interviewing and observation to collect information and, from that, you derive the requirements

Interviewing

- Formal or informal interviews with stakeholders are part of most RE processes.
- Types of interview
  - Closed interviews: based on pre-determined list of questions
  - Open interviews: where various issues are explored with stakeholders.
- Effective interviewing
  - Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.
  - Prompt the interviewee to get discussions going by using a springboard question, a requirements proposal, or by working together on a prototype system.

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 clear or think that it isn't worth articulating.

Ethnography

- Ethnography is an observational technique that can be used to understand operational processes and help derive requirements for software to support these processes.
- An analyst immerses himself or herself in the working environment where the system will be used.
- The day-to-day work is observed, and notes are made of the actual tasks in which participants are involved
- Ethnography can be combined with the development of a system prototype

Scenario

- Scenarios are real-life examples of how a system can be used.
- Scenarios are ways of capturing this kind of information.
- Stories are written as narrative text and present a high-level description of system use
- Scenarios are usually structured with specific information collected such as inputs and outputs.
- Scenarios capture the system, as viewed from the outside, e.g., by a user, using specific examples.

Scenario

- At its most general, a scenario may include:
  - A description of what the system and users expect when the scenario starts.
  - A description of the normal flow of events in the scenario.
  - A description of what can go wrong and how resulting problems can be handled.
  - Information about other activities that might be going on at the same time.
  - A description of the system state when the scenario ends.

Example Scenario: uploading photos in KidsTakePics

Uploading photos to KidsTakePics

Initial assumption: A user or a group of users have one or more digital photographs to be uploaded to the picture-sharing site. These photos are saved on either a tablet or a laptop computer. They have successfully logged on to KidsTakePics.

Normal: The user chooses to upload photos and is prompted to select the photos to be uploaded on the computer and to select the project name under which the photos will be stored. Users should also be given the option of inputting keywords that should be associated with each uploaded photo. Uploaded photos are named by creating a conjunction of the user name with the filename of the photo on the local computer.

On completion of the upload, the system automatically sends an email to the project moderator, asking them to check new content, and generates an on-screen message to the user that this checking has been done.

What can go wrong: No moderator is associated with the selected project. An email is automatically generated to the school administrator asking them to nominate a project moderator. Users should be informed of a possible delay in making their photos visible.

Photos with the same name have already been uploaded by the same user. The user should be asked if he or she wishes to re-upload the photos with the same name, rename the photos, or cancel the upload. If users choose to re-upload the photos, the originals are overwritten. If they choose to rename the photos, a new name is automatically generated by adding a number to the existing filename.

Other activities: The moderator may be logged on to the system and may approve photos as they are uploaded.

System state on completion: User is logged on. The selected photos have been uploaded and assigned a status "awaiting moderation." Photos are visible to the moderator and to the user who uploaded them.

4.4 Requirements Specification

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

Requirement Specifications

- Requirements specification is the process of writing down the user and system requirements in a requirements document.
- The user and system requirements should be clear, unambiguous, easy to understand, complete, and consistent
- User requirements are almost always written in natural language supplemented by appropriate diagrams and tables in the requirements document.
- System requirements may also be written in natural language, but other notations based on forms, graphical, or mathematical system models can also be used

Requirement Specifications

- Possible notations for writing system requirements

Notation	Description
Natural language sentences	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system. UML (unified modeling language) use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want, and they are reluctant to accept it as a system contract. (I discuss this approach, in Chapter 10, which covers system dependability.)

Requirement Specifications

- Structured Natural Language

Insulin Pump/Control Software/SRS/3.3.2

Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered.
Destination	Main control loop.
Action:	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered. (see Figure 4.14)
Requires	Two previous readings so that the rate of change of sugar level can be computed.
Precondition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Postcondition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

Graphical Notations: Use Cases

- In the Unified Modeling Language (UML), a use case diagram is used to summarize the details of your system's users (also known as actors) and their interactions with the system.
- An effective use case diagram can help your team discuss and represent:
  - Scenarios in which your system or application interacts with people, organizations, or external systems
  - Goals that your system or application helps those entities (known as actors) achieve
  - The scope of your system
- Use cases deal only in the functional requirements for a system.

Importance of Use Case Diagrams

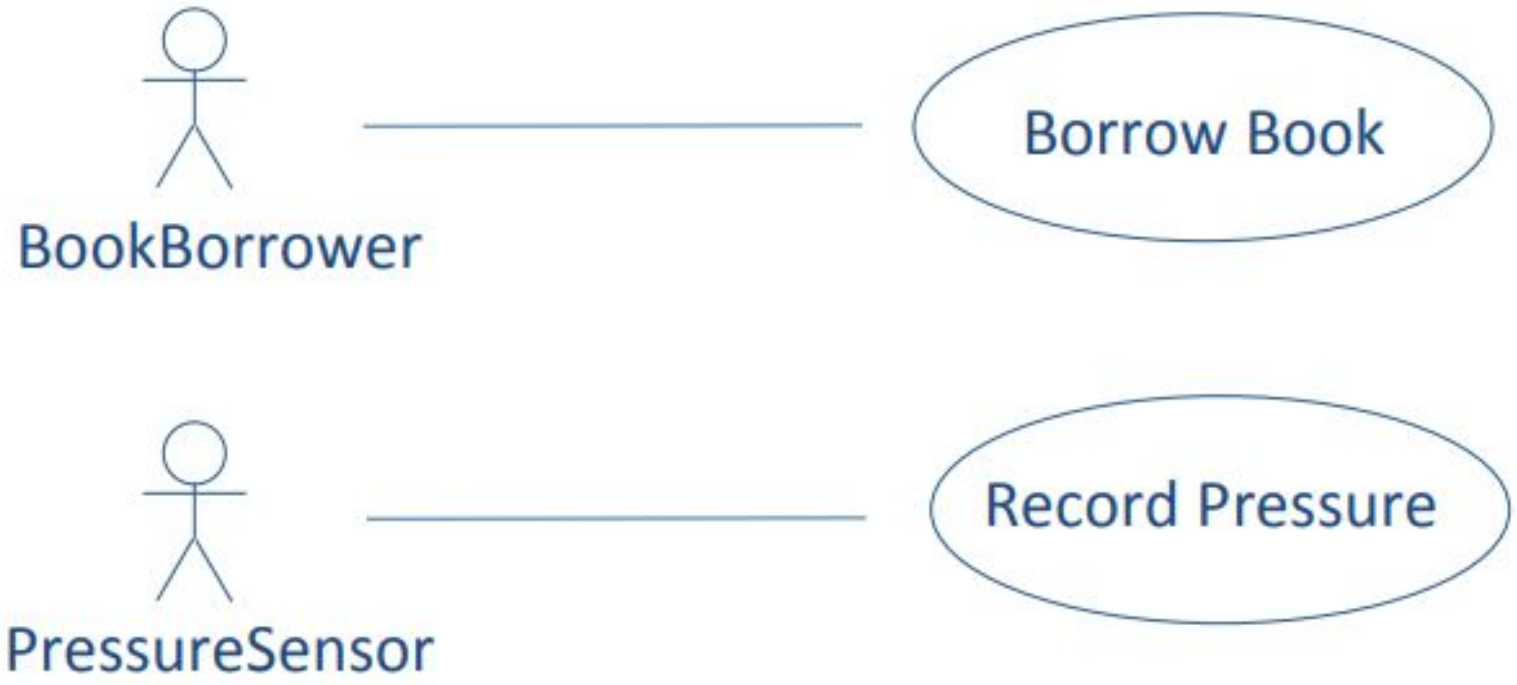
- To identify functions and how roles interact with them – The primary purpose of use case diagrams.
- For a high-level view of the system – Especially useful when presenting to managers or stakeholders. You can highlight the roles that interact with the system and the functionality provided by the system without going deep into inner workings of the system.
- To identify internal and external factors – This might sound simple but in large complex projects a system can be identified as an external role in another use case.

Objects of Use Case Diagram

Use case diagrams consist of 4 objects.

1. Actors
2. Use case
3. System
4. Relationships

Two Simple Use Case Diagrams

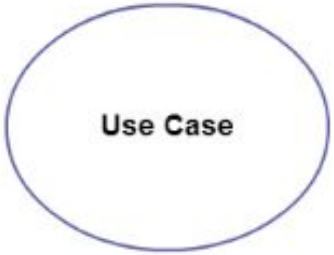


Actors

- Actor in a use case diagram is any entity that performs a role in one given system.
- This could be a person, organization or an external system and usually drawn like skeleton shown below.

Use Case

- A use case represents a function or an action within the system.
- It's drawn as an oval and named with the function.



System

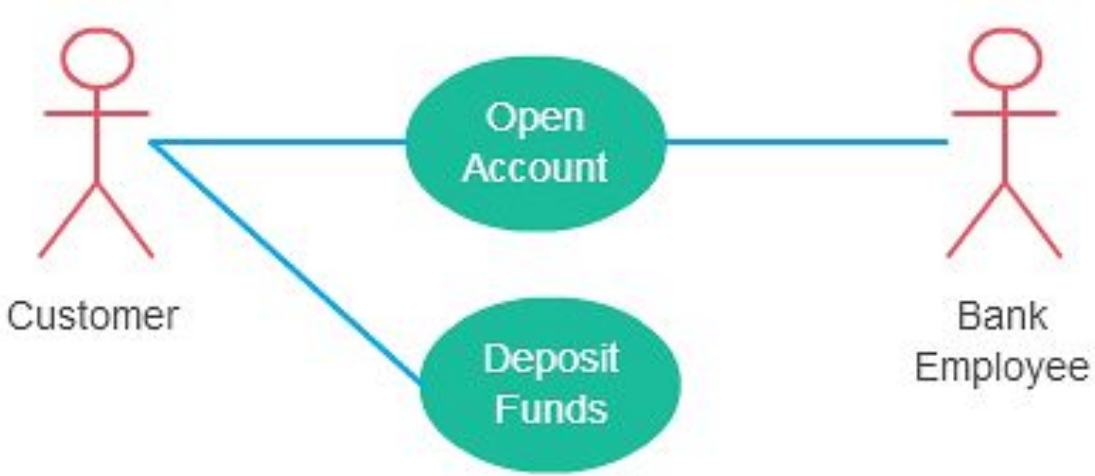
- The system is used to define the scope of the use case and drawn as a rectangle.
- This an optional element but useful when you're visualizing large systems.
- For example, you can create all the use cases and then use the system object to define the scope covered by your project.

Relationships

- There are four types of relationships in a use case diagram.
  1. Association between an actor and a use case
  2. Generalization of an actor
  3. Extend relationship between two use cases
  4. Include relationship between two use cases

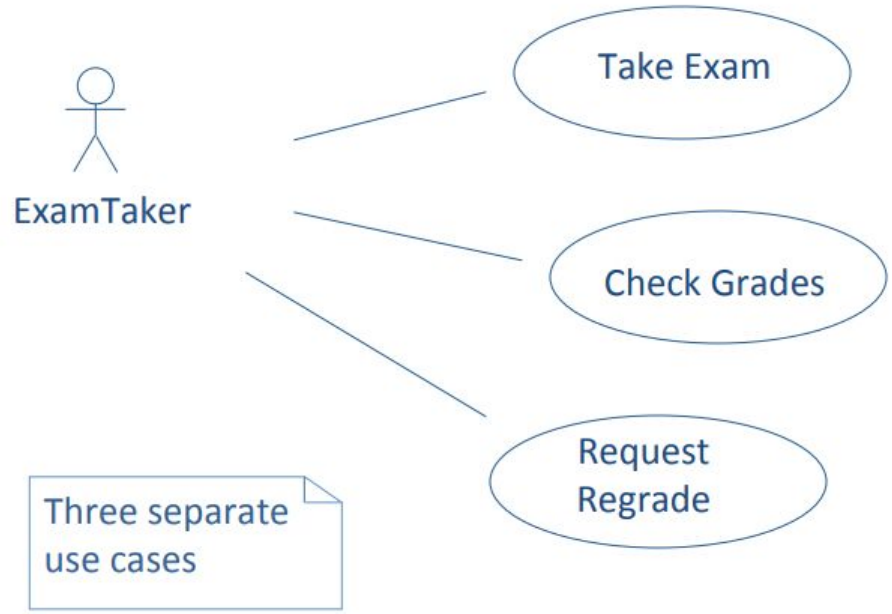
Relationships

- Association between an actor and a use case
  - An actor must be associated with at least one use case.
  - An actor can be associated with multiple use cases.
  - Multiple actors can be associated with a single use case



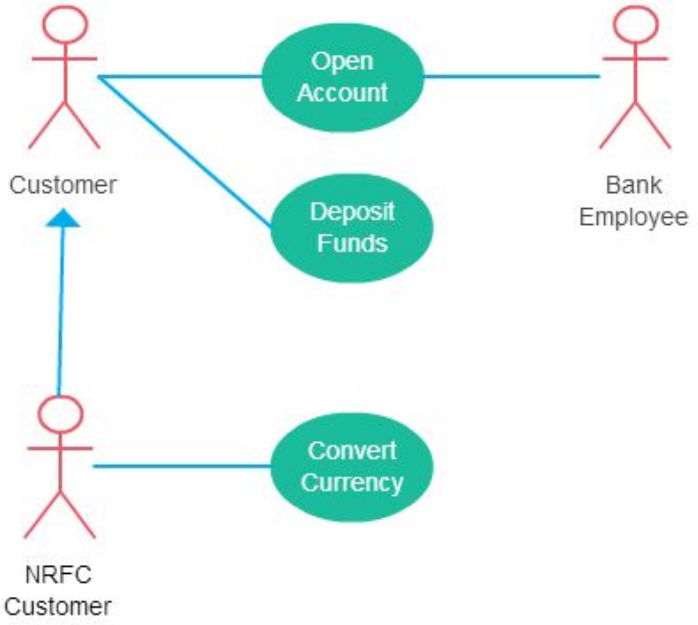
Relationships – Exam System

- Association between an actor and a use case



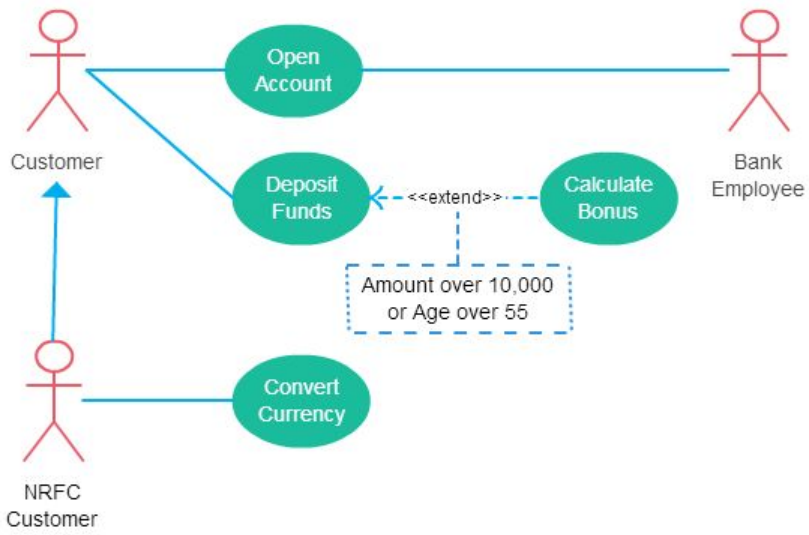
Relationships

- Generalization of an actor
  - Generalization of an actor means that one actor can inherit the role of the other actor.
  - The descendant inherits all the use cases of the ancestor.
  - The descendant has one or more use cases that are specific to that role.



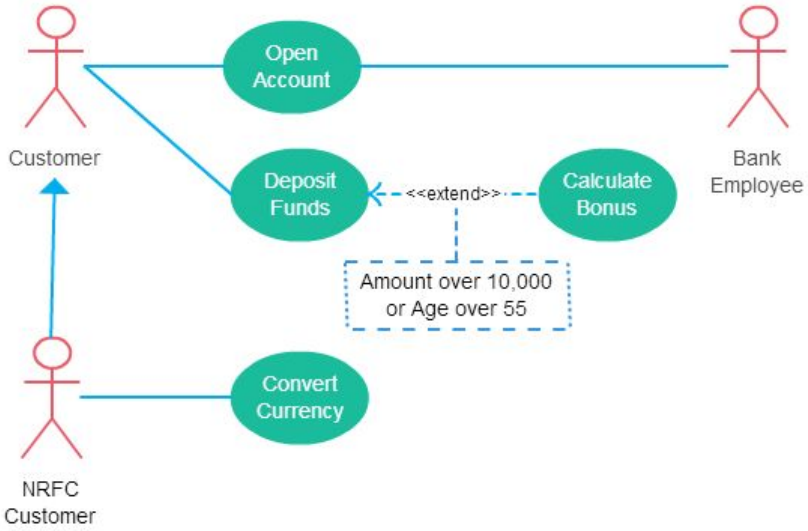
Relationships

- Extend relationship between two use cases
  - It extends the base use case and adds more functionality to the system



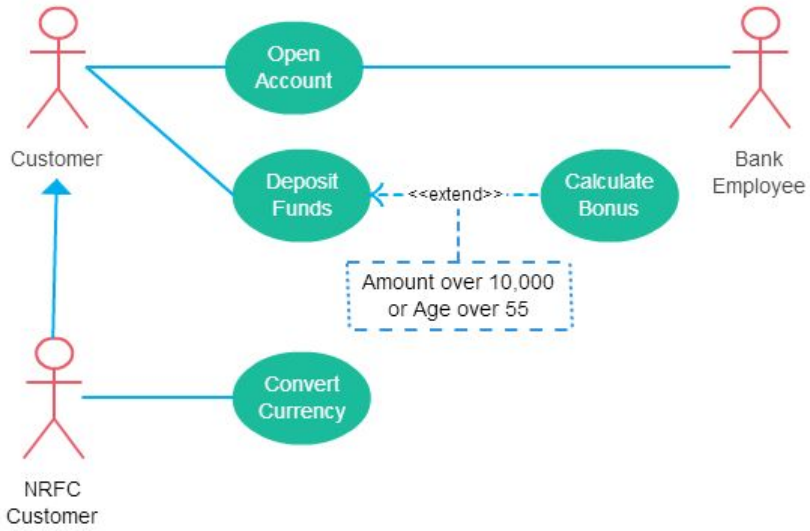
Relationships

- Extend relationship between two use cases
  - The extending use case is dependent on the extended (base) use case



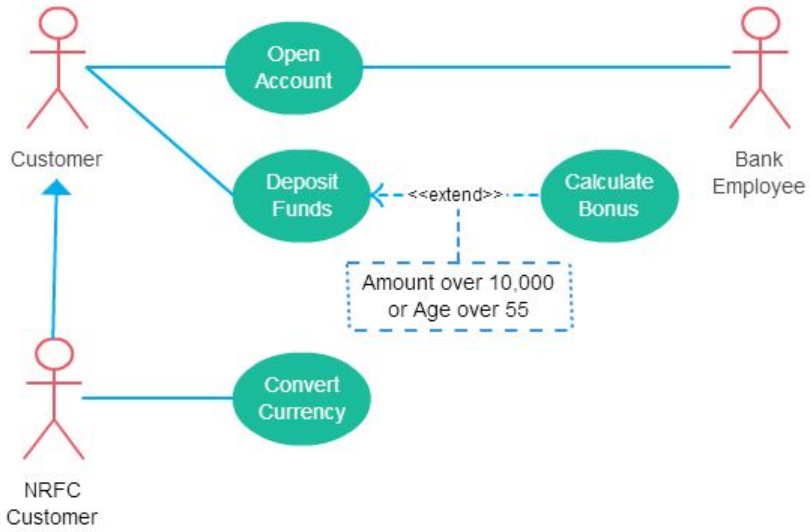
Relationships

- Extend relationship between two use cases
  - The extending use case is usually optional and can be triggered conditionally



Relationships

- Extend relationship between two use cases
  - The extended (base) use case must be meaningful on its own.

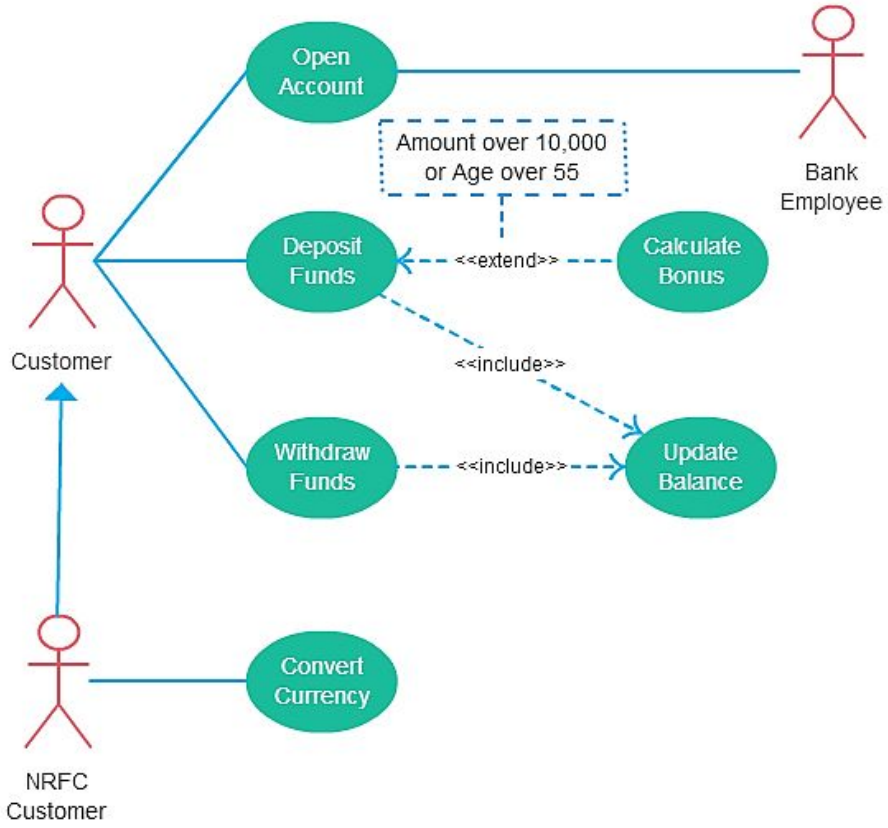


Relationships

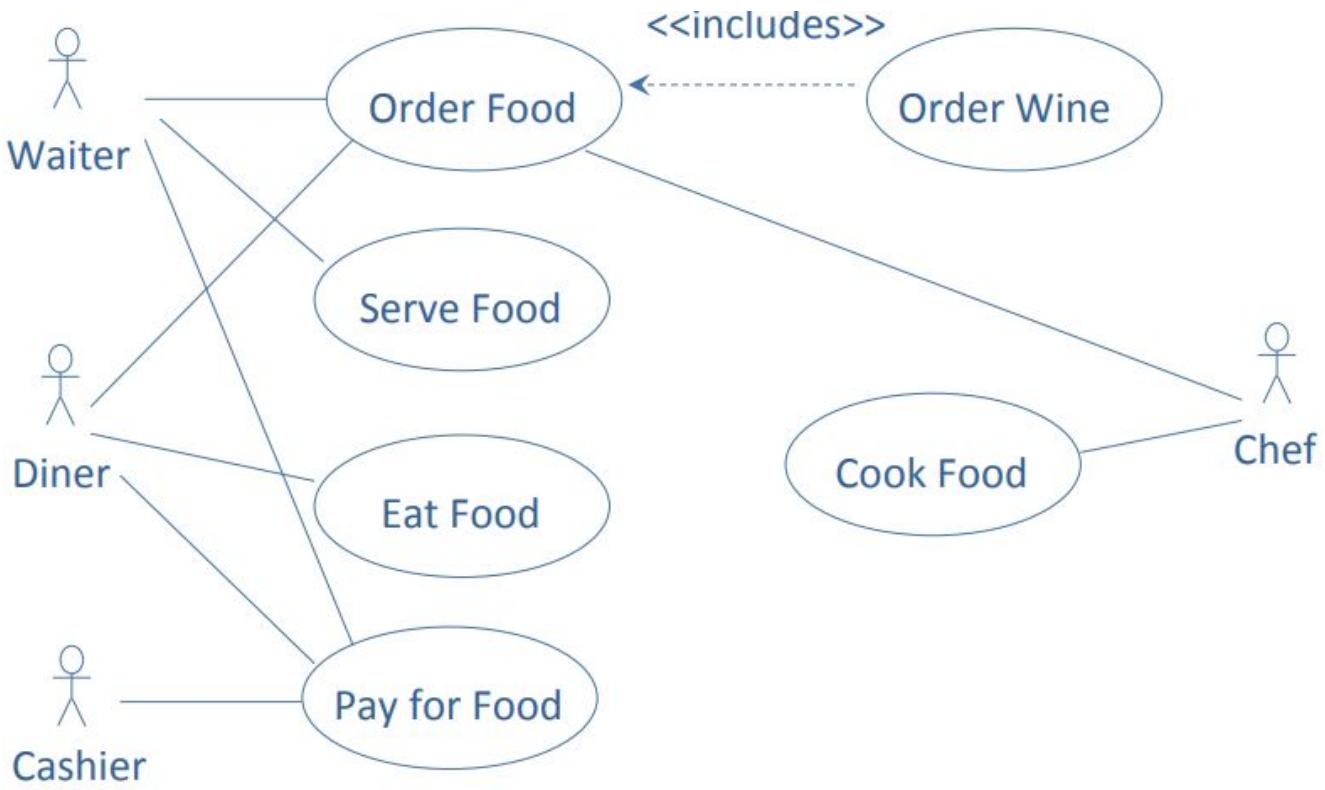
- Include relationship between two use cases
  - Include relationship show that the behavior of the included use case is part of the including (base) use case.
  - The main reason for this is to reuse the common actions across multiple use cases.

Relationships

- Include relationship between two use cases



Restaurant



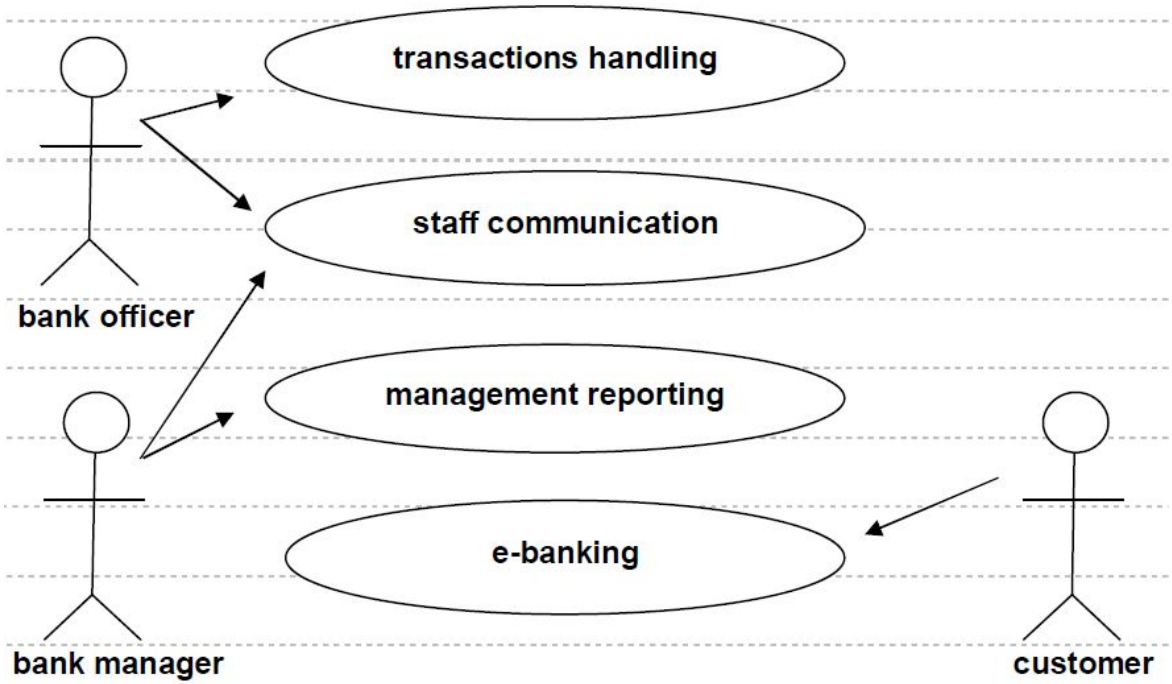
This restaurant example is based on a use case diagram from Wikipedia.

Activity

Suppose you are working in a software engineering team assigned the task of developing system for a bank. The system will be used for transactions handling, staff communication, management reporting and e-banking. Bank officers will be using the system for transactions handing and communication. The bank manager will be using the system to view management reports and communication. Bank customers will enjoy e-banking when the system is deployed.

Identify the actors of this system and draw the high level use case diagram for the system.

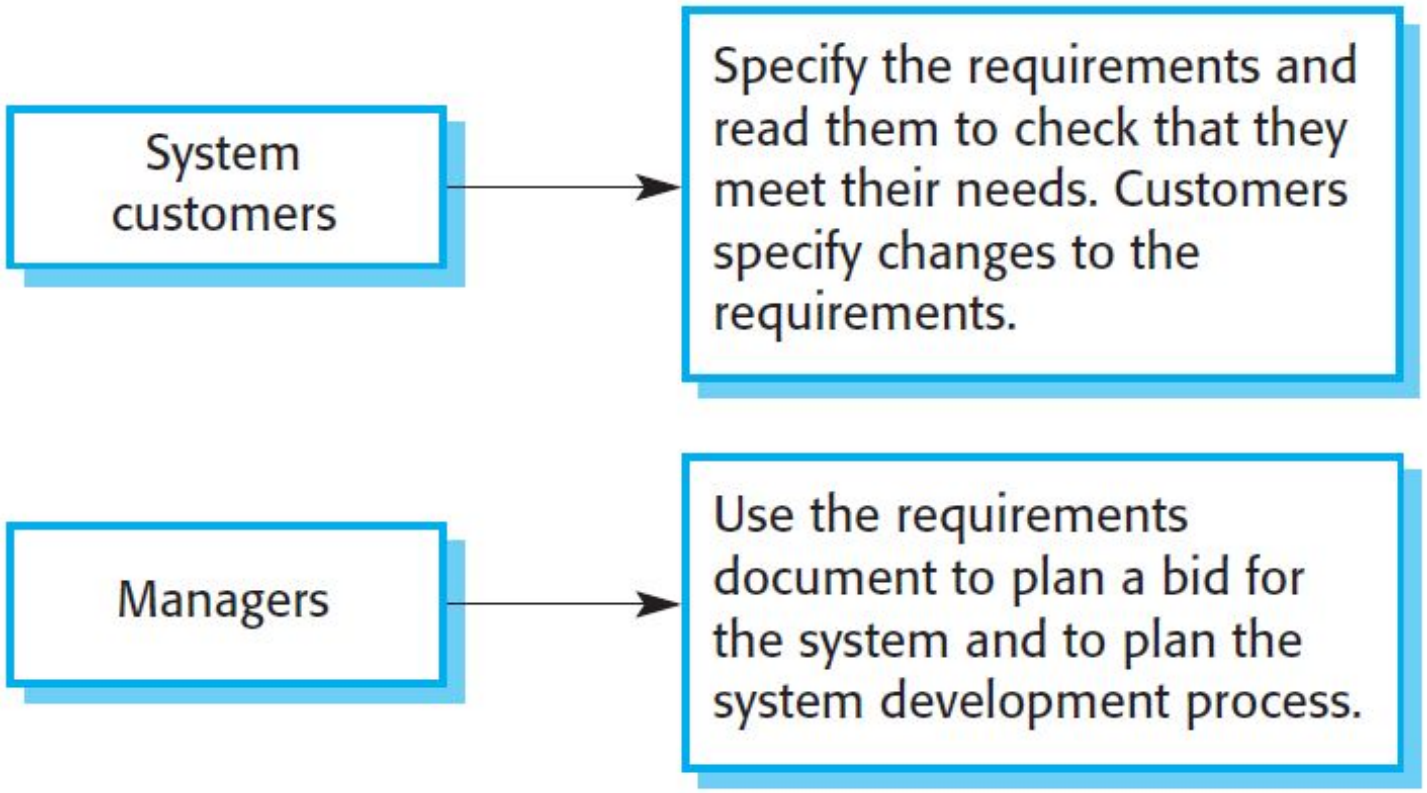
Activity: Answer



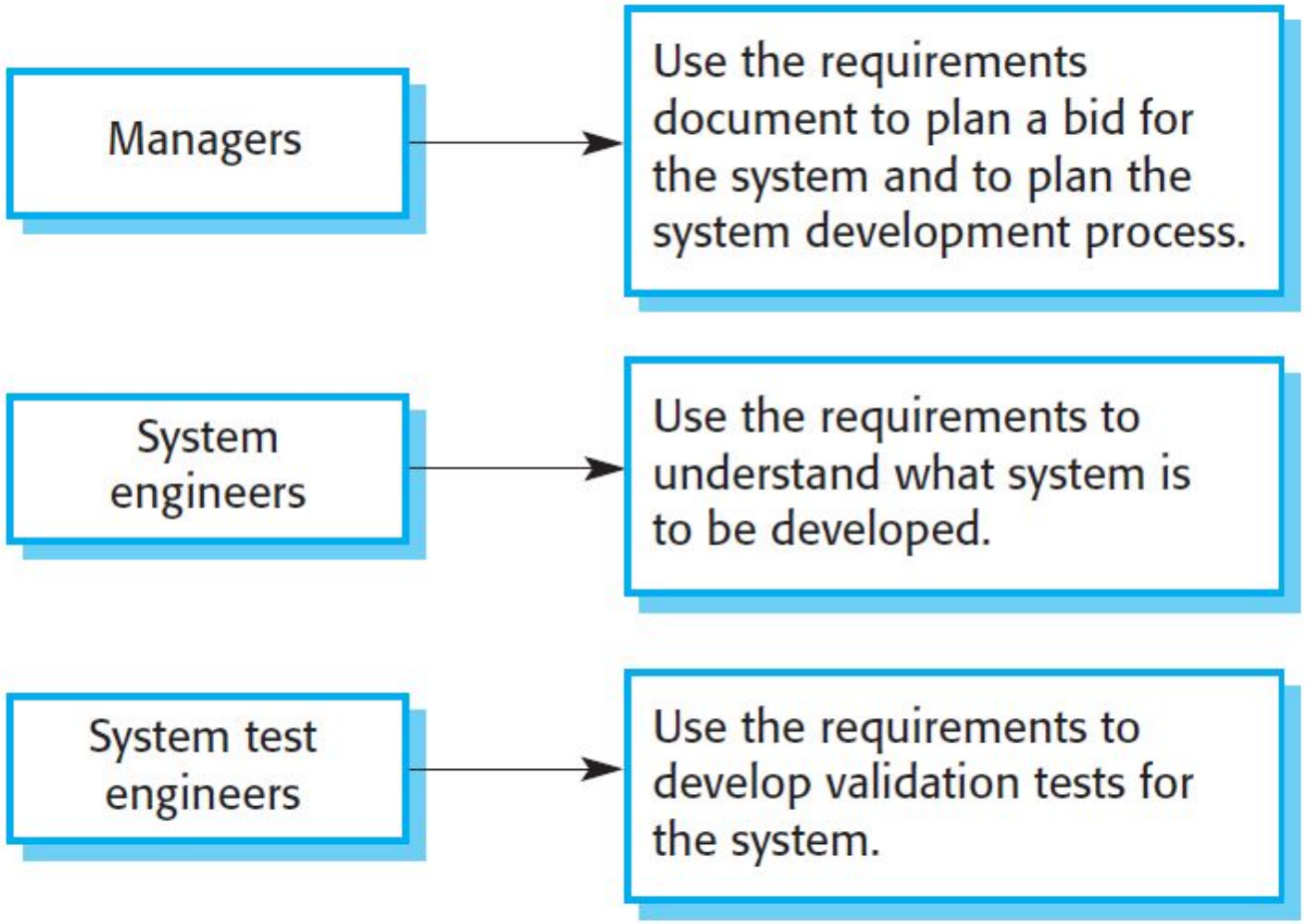
Software Requirements Document

- Sometimes called software requirements specification or SRS
- This is an official statement of what the system developers should implement.
- Includes both user and system requirements
- Requirements documents are essential when systems are outsourced for development, when different teams develop different parts of the system, and when a detailed analysis of the requirements is mandatory.

Users of a Requirements Documents



Users of a Requirements Documents



Software Requirement Specification

- The level of detail that you should include in a requirements document depends on the type of system that is being developed and the development process used.
- Critical systems need detailed requirements because safety and security have to be analyzed in detail to find possible requirements errors.
- When the system is to be developed by a separate company (e.g., through outsourcing), the system specifications need to be detailed and precise.

Structure of a SRS

- Preface
- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices

Structure of a SRS

Structure of a SRS

- Preface:
  - This defines the expected readership of the document and describe its version history, including a summary of the changes made in each version.
- Introduction:
  - This describes the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
- Glossary:
  - This defines the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
- User requirements definition:
  - describe the services provided for the user. description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
- System Architecture
  - presents a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

Structure of a SRS

- User requirements definition:
  - describe the services provided for the user. description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
- System Architecture
  - presents a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
- System requirements specification:
  - describes the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
- System models:
  - This chapter includes graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
- System evolution
  - describes the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on.
  - This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
- Appendices:
  - These provide detailed, specific information that is related to the application being developed—for example, hardware and database descriptions.

4.5 Requirements Validation

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

Requirements Validation

- Requirements validation is the process of checking that requirements define the system that the customer really wants.
- It overlaps with elicitation and analysis, as it is concerned with finding problems with the requirements.
- Requirements validation is critically important because errors in a requirements document can lead to extensive rework costs when these problems are discovered during development or after the system is in service.

Requirements Checking

- **Validity**
  - Does the system provide the functions which best support the customer's needs?
- **Consistency**
  - Are there any requirements conflicts?
- **Completeness**
  - Are all functions required by the customer included?
- **Realism**
  - Can the requirements be implemented given available budget and technology
- **Verifiability**
  - Can the requirements be checked?

Requirements Validation Techniques

- **Requirements reviews**
  - Systematic manual analysis of the requirements.
- **Prototyping**
  - Using an executable model of the system to check requirements.
- **Test-case generation**
  - Developing tests for requirements to check testability.

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?

4.6 Requirements Change

IT2206 - Fundamentals of Software Engineering

Level I - Semester 2

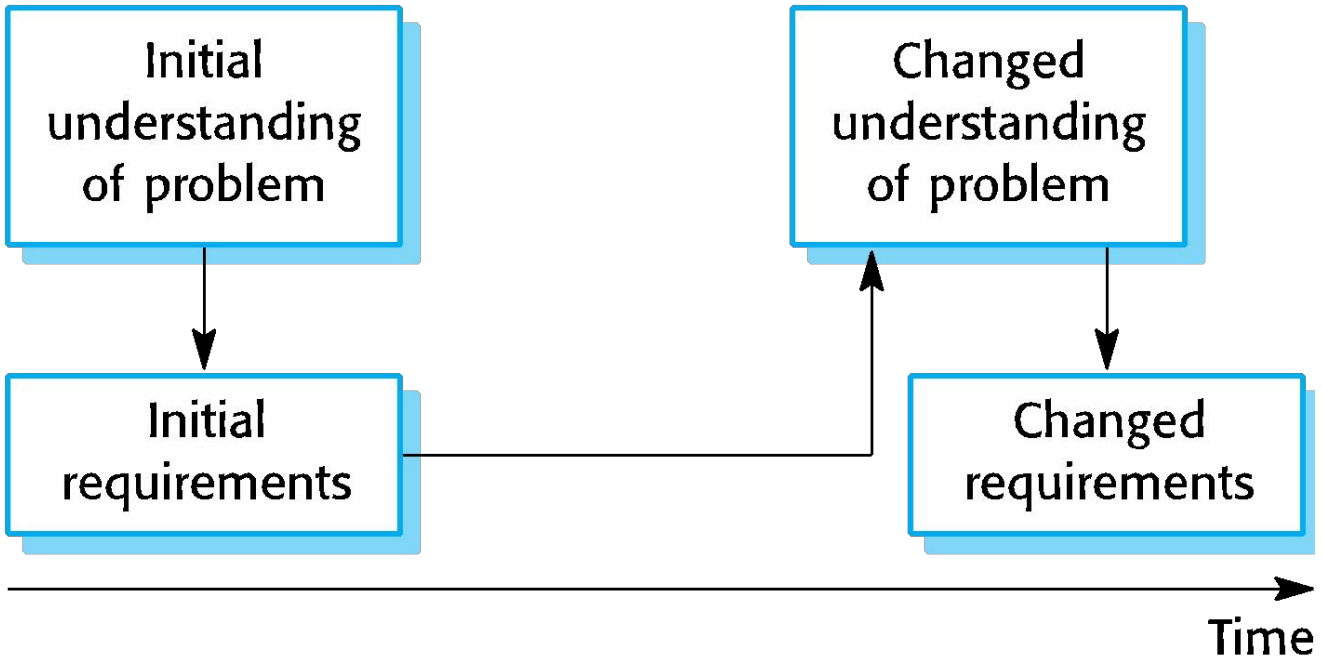
Changing Requirements

- The business and technical environment of the system always changes after installation.
  - New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by.
- The people who pay for a system and the users of that system are rarely the same people.
  - System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.

Changing Requirements

- Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.
  - The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed.

Requirements Evolution



Requirements Management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge as a system is being developed and after it has gone into use.
- You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.

Requirements Management Planning

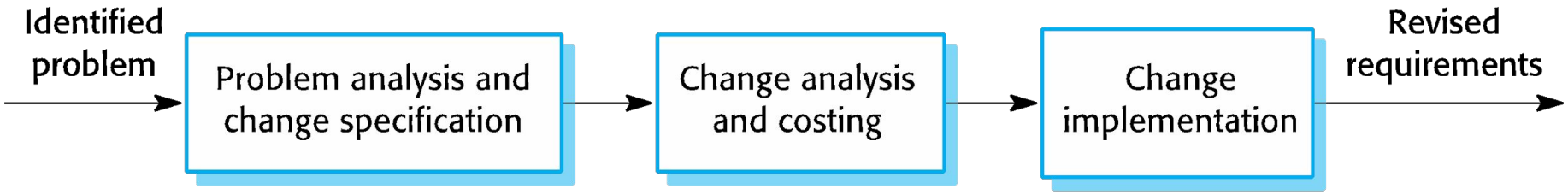
- Establishes the level of requirements management detail that is required.
- Requirements management decisions:
  - **Requirements identification**
    - Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
  - **A change management process**
    - This is the set of activities that assess the impact and cost of changes. I discuss this process in more detail in the following section.
  - **Traceability policies**
    - These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
  - **Tool support**
    - Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

Requirements Change Management

- Deciding if a requirements change should be accepted

  - **Problem analysis and change specification**
    - During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
  - **Change analysis and costing**
    - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
  - **Change implementation**
    - The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.

Requirements Change Management



Summary

- Requirements for a software system set out what the system should do and define constraints on its operation and implementation.
- Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.
- Non-functional requirements often constrain the system being developed and the development process being used.
- They often relate to the emergent properties of the system and therefore apply to the system as a whole.
- The requirements engineering process is an iterative process that includes requirements elicitation, specification and validation.
- Requirements elicitation is an iterative process that can be represented as a spiral of activities – requirements discovery, requirements classification and organization, requirements negotiation and requirements documentation.

Summary

- You can use a range of techniques for requirements elicitation including interviews and ethnography. User stories and scenarios may be used to facilitate discussions.
- Requirements specification is the process of formally documenting the user and system requirements and creating a software requirements document.
- The software requirements document is an agreed statement of the system requirements. It should be organized so that both system customers and software developers can use it.
- Requirements validation is the process of checking the requirements for validity, consistency, completeness, realism and verifiability.
- Business, organizational and technical changes inevitably lead to changes to the requirements for a software system. Requirements management is the process of managing and controlling these changes.