Unit 3

Software requirement Analysis and Specification

Requirement Definition

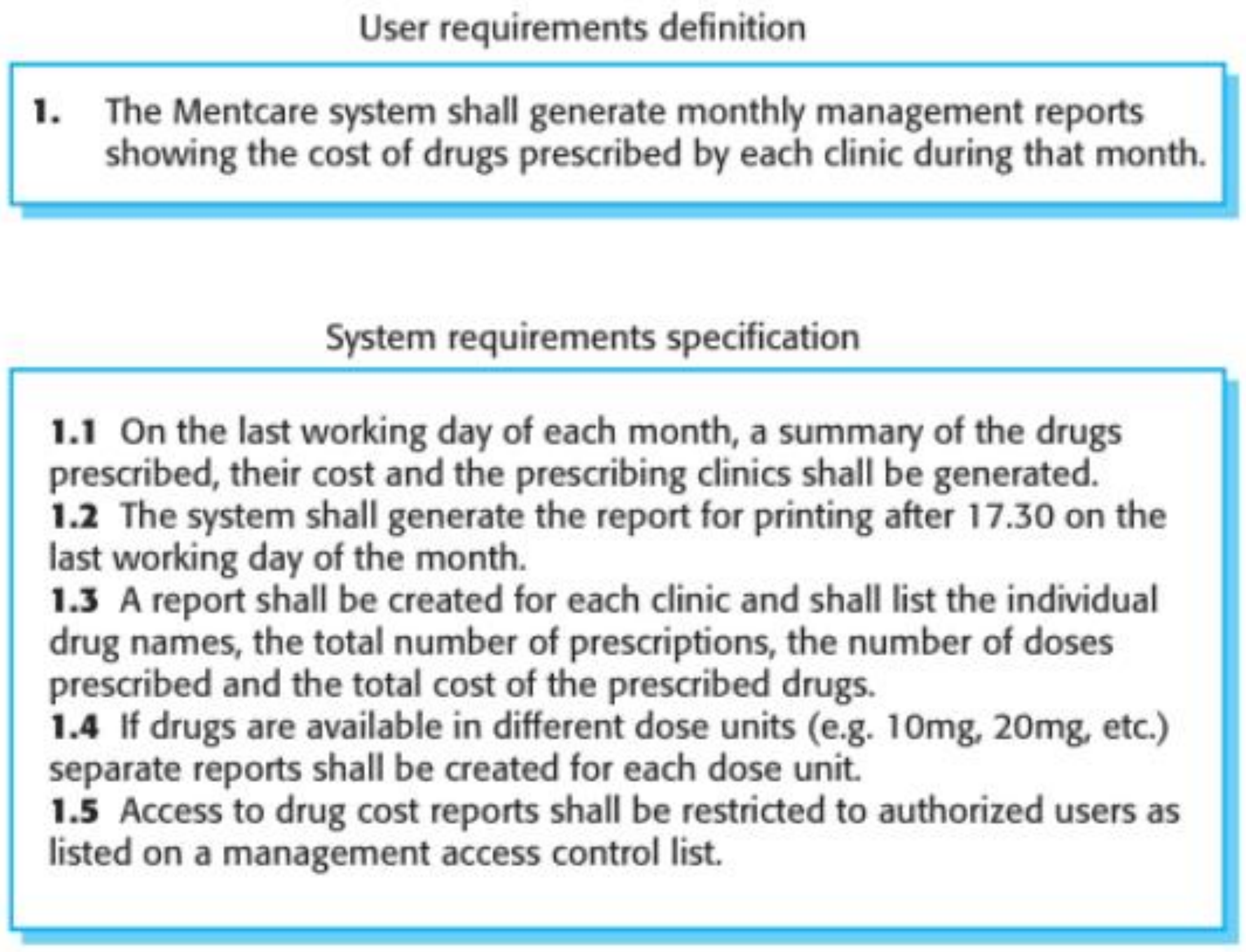
- The requirement for a system are the description of what the system should do, the service or services that it provides and the constraints on its operation. The IEEE Standard Glossary of Software Engineering Terminology defines a requirement as:
  - ***A condition or capability needed by a user to solve a problem or achieve an objective.***
  - A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
- Requirements are the descriptions and specification of the system.
- The software requirements are description of features and functionalities of the target system.
- Requirements convey the expectations of users from the software product.
- The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.
- A requirement may range from a high level abstract statement of a service or of a system constraint to a detailed mathematical functional specification of the system.
- 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.
- The process of finding out, analyzing and documenting and checking these services and constraints is called **requirement engineering**.

Requirements

- User requirement
  - It is the statement, in natural language and/or diagrams of what services the system is expected to provide and the constraints under which it must operate.
  - It represents high level abstract requirements.
- System requirement
  - It represents the detailed description of what system the system should do.
  - It sets out the system’s functions, services and operational constraints in detail.
  - It should be precise. It should define exactly, what is to be implemented.
- Software Specification:
  - A detail software description which can serve as a basis for a design or implementation. It is written specially for developers.

User requirements

- Should describe functional and non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge
- User requirements are defined using natural language, tables and diagram
- It is the statement, in natural language and/or diagrams of what services the system is expected to provide and the constraints under which it must operate.
- It represents high level abstract requirements.



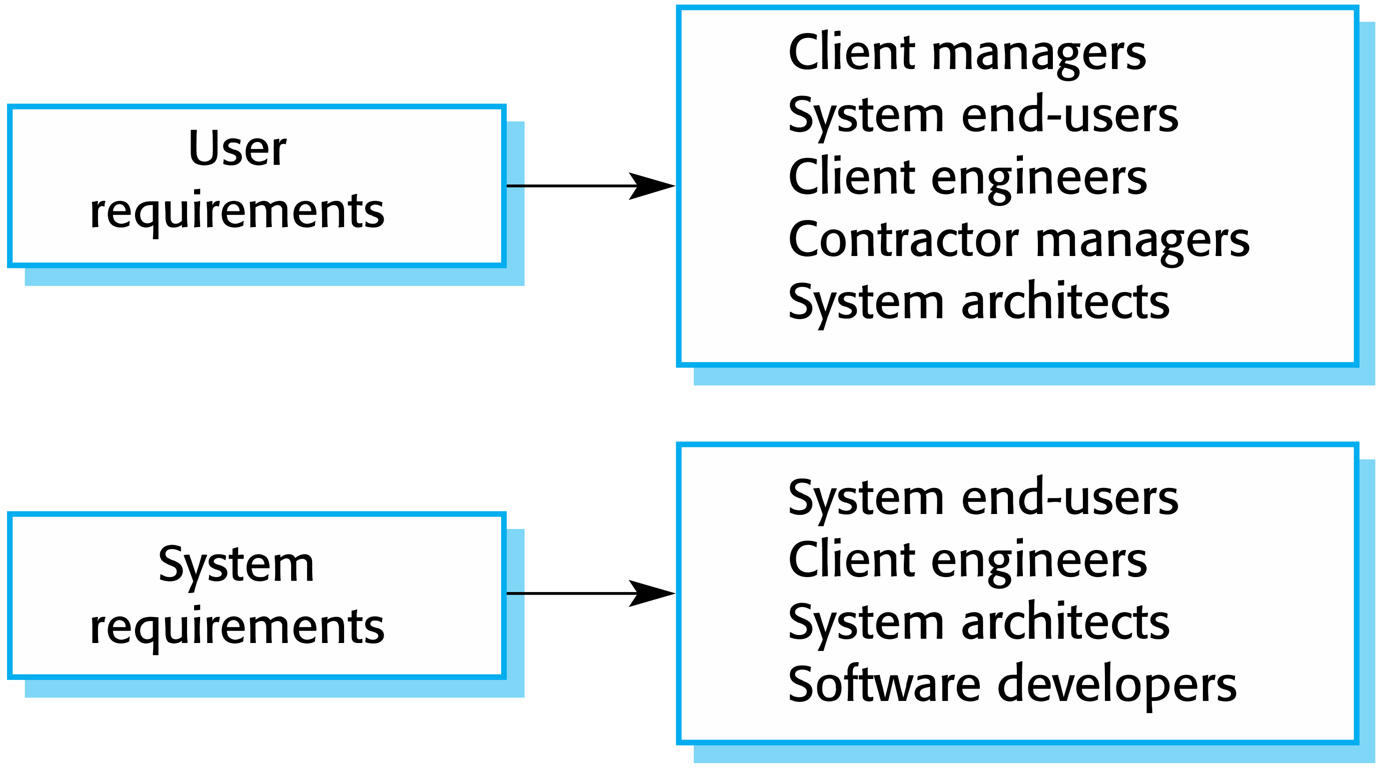
User requirements definition

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.

Readers of different types of requirements

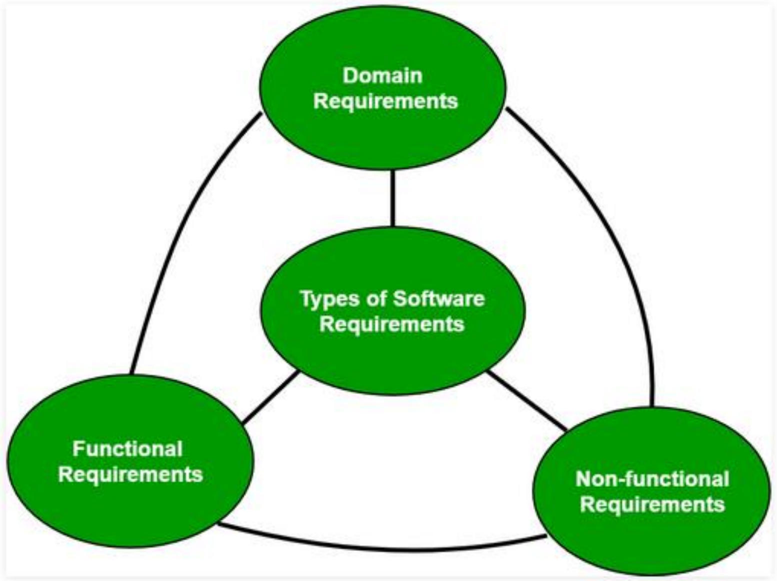


Difficulties in requirement Analysis

- Lack of understanding the domain or real problem
- Requirement changes rapidly
- Attempting to do too much
- Hard to reconcile conflicting set of requirements
- Hard to state requirements precisely(Business,process technology always changes)
- Problem in Natural Language understanding:exists ambiguity,lack of modularization,over flexibility etc.

Software Requirements and its type

- **Software Requirements**
  - The software requirements are description of features and functionalities of the target system.
  - Requirements convey the expectations of users from the software product.
  - The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.
- **Types of Software Requirements**
  - Functional Requirements
  - Non-Functional Requirements
  - Domain Requirements



Functional requirement

- It is the statement of service, the system should provide, condition that the system should react to particular inputs and the system operations that will be used in particular situations.
- These are the requirements that the end user specifically demands as basic facilities that the system should offer.
- All these functionalities need to be necessarily incorporated into the system as a part of the contract.
- These are represented or stated in the form of input to be given to the system, the operation performed and the output expected.
- In some cases, the functional requirements may also explicitly state what the system should do.
- This requirement depend on the type of s/w being developed, the expected users of the s/w and general approach taken by the organization when writing requirements.
- 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.
- There are many ways of expressing functional requirements e.g., natural language, a structured or formatted language with no rigorous syntax and formal specification language with proper syntax.
- Some typical functional requirements are :
    - Business Rules
    - Transaction corrections, adjustments and cancellations
    - Administrative functions
    - External Interfaces
    - Certification Requirements
    - Reporting Requirements
    - Historical Data

Functional requirements examples

- Authentication of a user when he/she tries to log into the system.
- System shutdown in the case of a cyber attack.
- Verification email is sent to user whenever he/she registers for the first time on some software system.
- 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.
- The software automatically validates customers against the Contact Management System
- The Sales system should allow users to record customers sales
- Only Managerial level employees have the right to view revenue data.
- User must input username and password in order to login the system.

Advantages of functional requirements

- Helps you to check whether the application is providing all the functionalities that were mentioned in the functional requirement of that application
- A functional requirement document helps you to define the functionality of a system or one of its subsystems.
- Functional requirements along with requirement analysis help identify missing requirements. They help clearly define the expected system service and behavior.
- Errors caught in the Functional requirement gathering stage are the cheapest to fix.
- Support user goals, tasks, or activities for easy project management
- Functional requirement can be expressed in Use Case form or user story as they exhibit externally visible functional behavior.

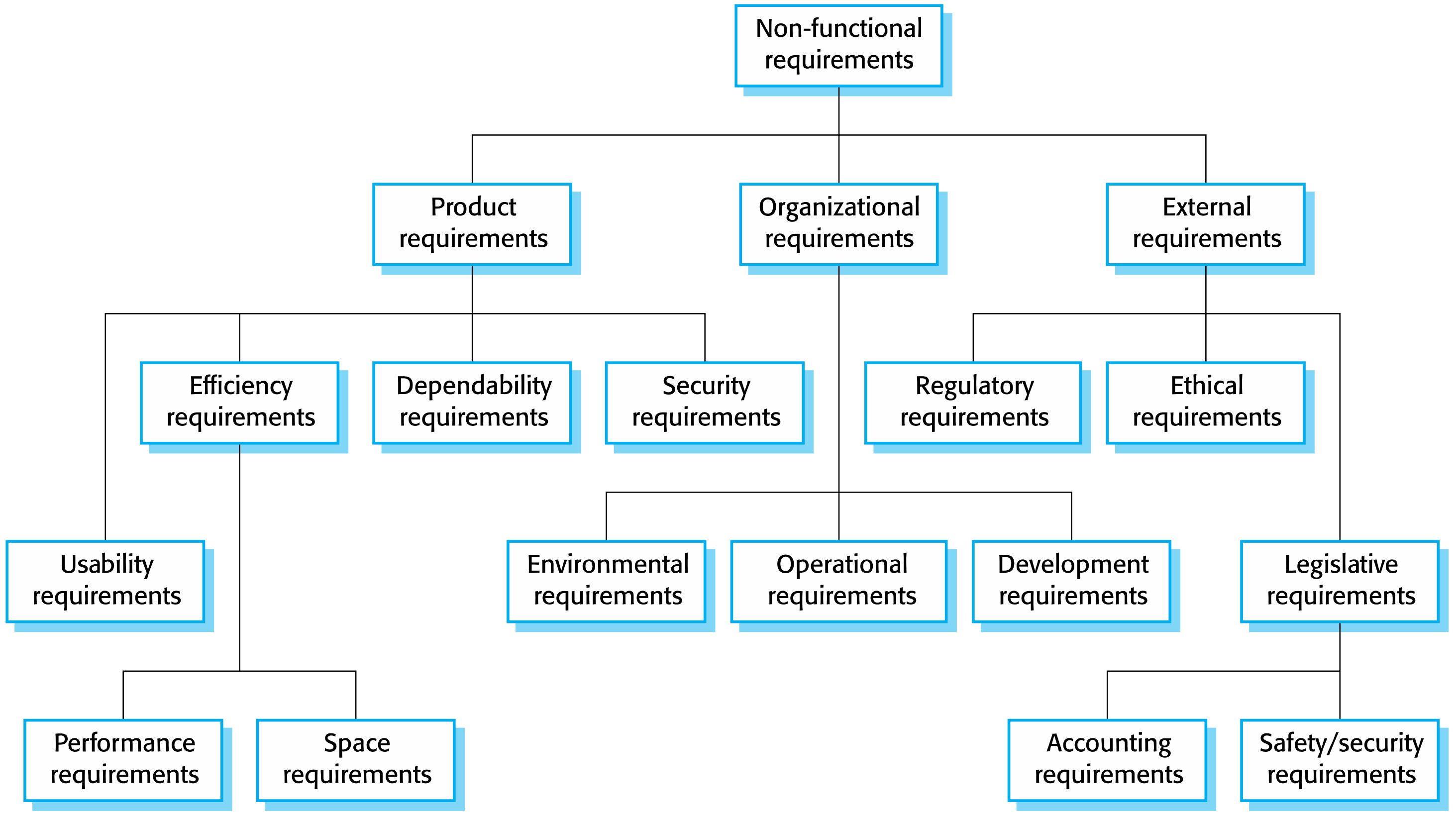
Non-functional requirements

- It is the constraints on the services or functions offered by the system.
- It include timing constraints, constraints on the development process and standards.
- It may relate to emergent system properties, such as reliability, response time and store occupancy.
- It may define constraints on the system such as the capabilities of I/O device and the data representation used in system interfaces.
- It is more critical than functional requirements.
- Some typical non functional requirements are:
    - Performance – for example Response Time, Throughput, Utilization, Static Volumetric
    - Scalability, availability , reliability
    - Capacity
    - Recoverability, Maintainability, Security ,Usability,Interoperability
    - Regulatory
    - Manageability
    - Data Integrity

Non-Functional requirements examples

- Emails should be sent with a latency of no greater than 12 hours.
- Each request should be processed within 10 seconds.
- The site should load in 3 seconds when the number of simultaneous users are > 10000
- Users must change the initially assigned login password immediately after the first successful login. Moreover, the initial should never be reused.
- Employees never allowed to update their salary information. Such attempt should be reported to the security administrator.
- Every unsuccessful attempt by a user to access an item of data shall be recorded on an audit trail.
- A website should be capable enough to handle 20 million users with affecting its performance
- The software should be portable. So moving from one OS to other OS does not create any problem.
- Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audited.

Types of nonfunctional requirement



Types of non functional requirements

- **Product requirement**: It specifies product behavior. E.g. performance, reliability, portability, usability requirements.
- **Organizational requirement**: It specifies requirements related to organizational policies and procedures. E.g. process standard used, implementation requirements etc.
- **External requirement**: It specifies the requirements which arise from factors which are external to the system and its development process. E.g. interoperability, legislative requirements.

Advantages of Non-functional requirements

- Nonfunctional requirements ensure the software system follow legal and compliance rules.
- They ensure the reliability, availability, and performance of the software system
- They ensure good user experience and ease of operating the software.
- They help in formulating security policy of the software system.

Functional vs Non Functional Requirements

Functional Requirements

- A functional requirement defines a system or its component.
- It specifies "What should the software system do?"
- Functional requirement is specified by User.
- It is mandatory
- It is captured in use case.
- Defined at a component level
- Helps you verify the functionality of the software.
- Functional Testing like System, Integration, End to End, API testing, etc are done.
- Usually easy to define.
- **Example:**
  - **1)** Authentication of user whenever he/she logs into the system.
  - **2)** System shutdown in case of a cyber attack.
  - **3)** A Verification email is sent to user whenever he/she registers for the first time on some software system.

Non-Functional Requirements

- A non-functional requirement defines the quality attribute of a software system.
- It places constraints on "How should the software system fulfill the functional requirements?"
- Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
- It is not mandatory in some cases
- It is captured as a quality attribute
- Applied to a system as a whole
- Helps you to verify the performance of the software.
- Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
- Usually more difficult to define.
- **Example**
  - **1)** Emails should be sent with a latency of no greater than 12 hours from such an activity.
  - **2)** The processing of each request should be done within 10 seconds
  - **3)** The site should load in 3 seconds when the number of simultaneous users are > 10000

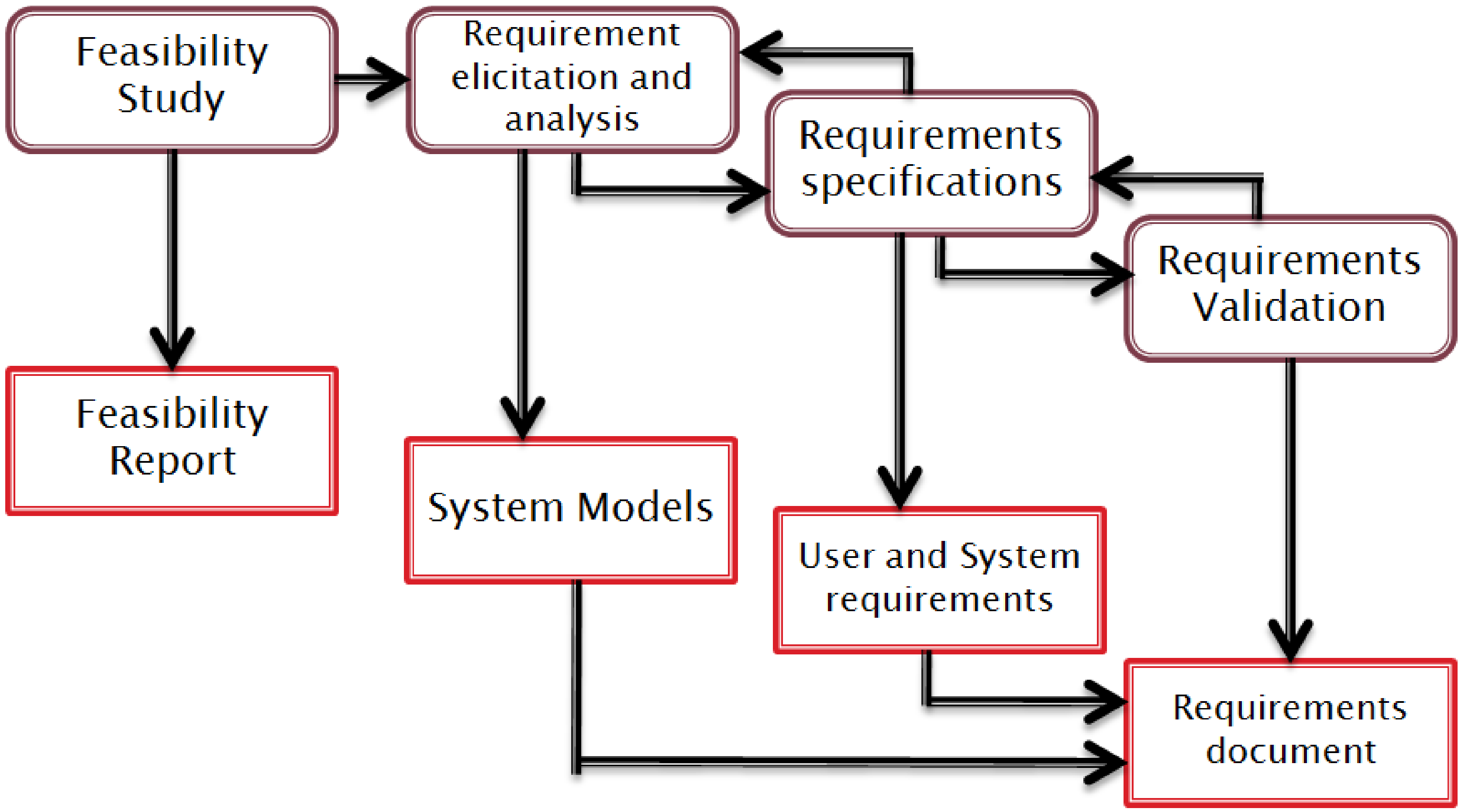
Domain Requirements

- Domain requirements are the requirements which are characteristic of a particular category or domain of projects.
- The basic functions that a system of a specific domain must necessarily exhibit come under this category.
- For instance, in an academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement.
- These requirements are therefore identified from that domain model and are not user specific.
- If domain requirements are not satisfied, the system may be unworkable.
- **Domain requirements Problems**
  - Understandability
    - Requirements are expressed in the language of application domain which may not me understood by the system developers
  - Implicitness
    - Domain specialist make the requirement implicit that is the requirements are understood by domain specialist only

Requirement Engineering Process

- Requirement Engineering is systematic use of proven principles, techniques and languages tools for the cost effective analysis, documentation and on going evaluation of user's need and the specification of the external behavior of the system to satisfy those user needs.
- It provides the appropriate mechanism for understanding what the customers want, analyzing the need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification and managing the requirements as they are transformed into a operational system.
- It is the process to create and maintain system requirement document.
- It is concerned with:
  - ❖Whether the system is useful to business (feasibility study)
  - ❖Discovering requirements (elicitation and analysis)
  - ❖Converting these requirement into some standard form (specification)
  - ❖Checking that the requirements actually define the system the customer wants (validation).

Requirement Engineering Process



Feasibility Studies

- An estimate is made of whether the identified user needs may be satisfied using current s/w and h/w technologies.
- The study considers whether the proposed system will be cost effective from the business point of view and whether it can be developed within existing budget or not.
- It also includes legal feasibility, social feasibility, schedule feasibility, organizational feasibility etc.
- A feasibility study is a study made before committing to a project.
- Feasibility study is carried out based on many purposes to analyze whether software product will be right in terms of development, implantation, contribution of project to the organization etc
- A feasibility study leads to a decision:
  - go ahead
  - do not go ahead
  - think again In production projects,
- the feasibility study often leads to a budget request.
- A feasibility study may be in the form of a proposal.

- Types of Feasibility Study
  - **Technical Feasibility**
    - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
  - **Operational Feasibility**
    - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
  - **Economic Feasibility**
    - Economic feasibility decides whether the necessary software can generate financial profits for an organization.
  - **Legal Feasibility**
    - Legal Feasibility study project is analyzed in legality point of view like analyzing barriers of legal implementation of project, data protection acts or social media laws, project certificate, license, copyright etc. to conform legal and ethical requirements.
  - **Schedule Feasibility**
    - Schedule Feasibility Study mainly timelines/deadlines is analyzed for proposed project which includes how many times teams will take to complete final project which has a great impact on the organization as purpose of project may fail if it can’t be completed on time.

- **Feasibility Study Process :**
  The below steps are carried out during entire feasibility analysis.
  - Information assessment
  - Information collection
  - Report writing
  - General information

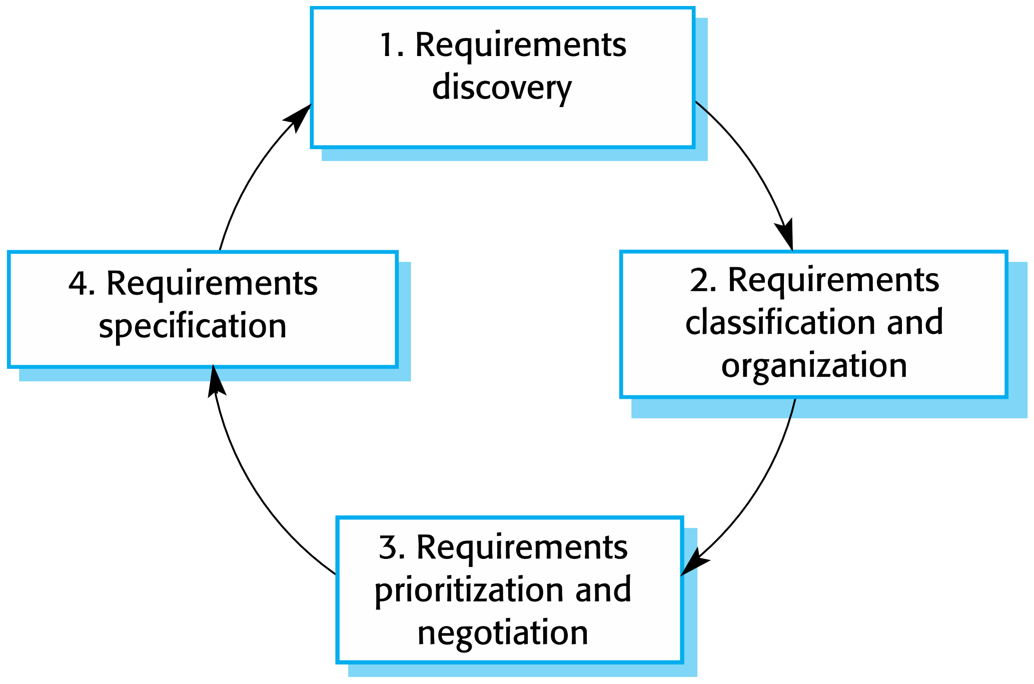
Requirement elicitation and analysis

- This is the process of deriving the system requirements through observations of existing systems, discussions with potential users and processes, task analysis and so on.
- It involves technical staff working with customers to find out about the application domain, the services that the system should provide and system's operational constraints.
- Sometimes called requirements elicitation or requirements discovery.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*
- 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.
- Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.
- Various Stages include in Requirement elicitation are:
  - Requirements discovery,
  - Requirements classification and organization,
  - Requirements prioritization and negotiation,
  - Requirements specification.

Requirement Ellicitation and analysis process

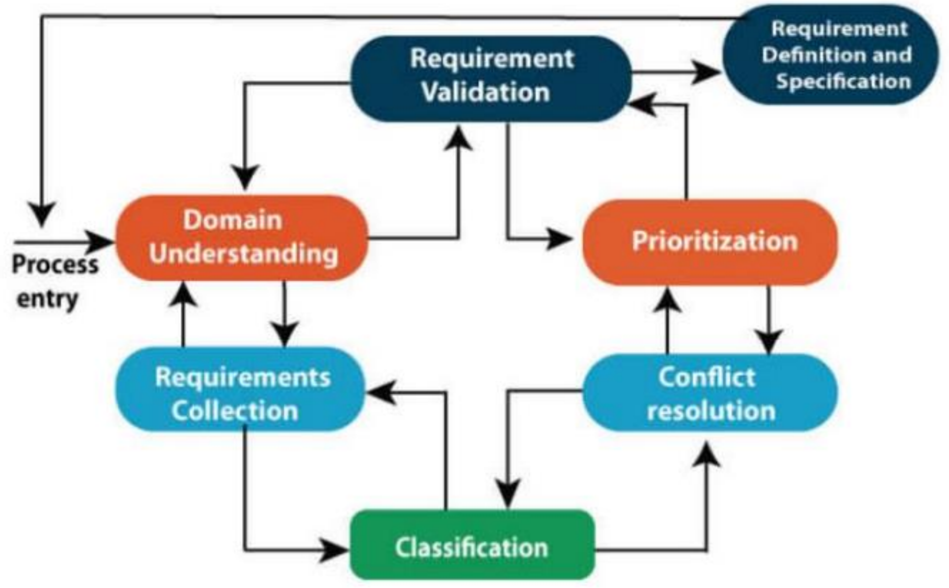
- Various stages involves are:
  - Requirements discovery,
  - Requirements classification and organization,
  - Requirements prioritization and negotiation,
  - Requirements specification.

1. **Requirements discovery and understanding** This is the process of interacting with stakeholders of the system to discover their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity.
2. **Requirements classification and organization** This activity takes the unstructured collection of requirements, groups related requirements and organizes them into coherent clusters.
3. **Requirements prioritization and negotiation** Inevitably, when multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation. Usually, stakeholders have to meet to resolve differences and agree on compromise requirements.
4. **Requirements documentation** The requirements are documented and input into the next round of the spiral. An early draft of the software requirements documents may be produced at this stage, or the requirements may simply be maintained informally on whiteboards, wikis, or other shared spaces.


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

- Activities involved in requirement analysis process are:
    - Domain understanding :understand the problem domain clearly by interaction and communication with different stakeholders
    - Classification
    - Conflict resolution
    - Prioritization
    - Requirements validation and checking

Elicitation and Analysis Process



- Problems in Requirement Ellicitation
    - Stakeholders don't know what they really want.
    - Stakeholders express requirements in their own terms.
    - Different stakeholders 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 may change.

Techniques of Requirement Elicitation/Solicitation or Requirement Discovery

- **Three methods:**
  - **Traditional methods**: Interviews, Questionnaire, Observation, Document Analysis,Viewpoints, Scenarios,Ethnography etc.
  - **Modern methods**: Joint Application Development(JAD),Facilitated Application Specification Technique(FAST),Prototyping, Brainstorming, Use Case approach
  - **Radical Methods**: Business Process Reengineering(BPR)

Interviews

- Discover issues with the current system and needs for the future
- requirements engineering teams put the questions to the stakeholder about the system that's currently used, and the system to be developed, and hence they can gather the requirements from the answers.
- Gather facts, opinions and speculations
- Observe body language and emotions
- Interview can be of one to one (direct) and indirect
- Questions can be of
  - Open ended: generic and unstructured questions
  - Close ended: predefined and structured questions

- Guidelines for making interview
  - Plan
    - Checklist
    - Appointment
  - Be neutral
  - Listen
  - Seek a diverse view

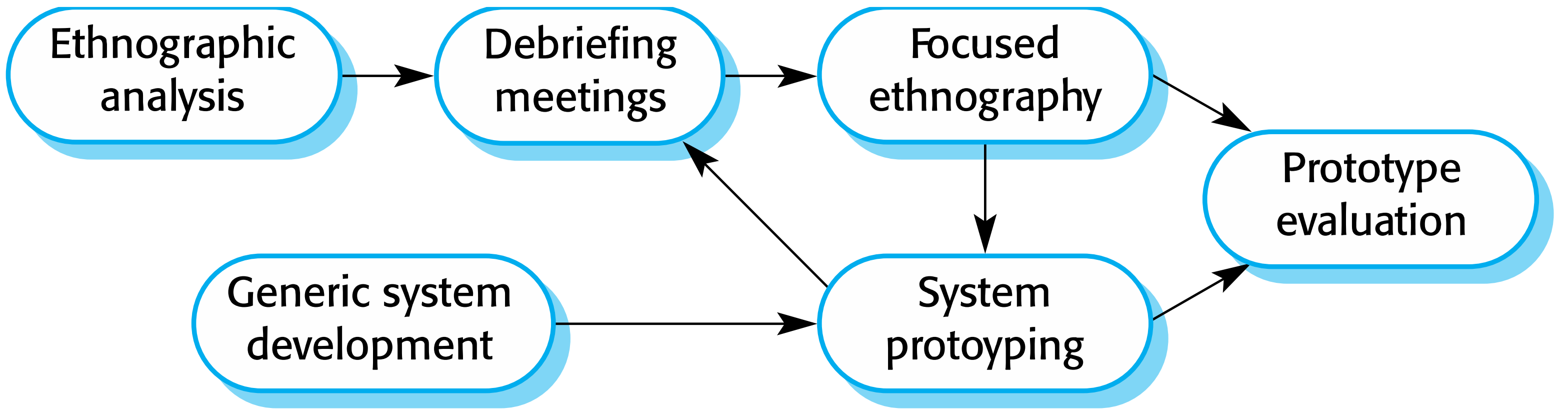
- **Basic Rules of Interview:**
  - The overall purpose of performing the interviews should be clear.
  - Identify the interviewees in advance.
  - Interview goals should be communicated to the interviewee.
  - Interview questions should be prepared before the interview.
  - The location of the interview should be predefined.
  - The time limit should be described.
  - The interviewer should organize the information and confirm the results with the interviewees as soon as possible after the interview.
- **Benifits**
  - Interactive discussion with stakeholders.
  - The immediate follow-up to ensure the interviewer’s understanding.
  - Encourage participation and build relationships by establishing rapport with the stakeholder.
- **Drawbacks**
  - Time is required to plan and conduct interviews.
  - Commitment is required from all the participants.
  - Sometimes training is required to conduct effective interviews.
  - 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.

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.
- The value of ethnography is that it helps discover implicit system requirements that reflect the actual ways that people work, rather than the formal processes defined by the organization.
- Scope of Ethnography
  - Requirements derived from the way in which people actually work, rather than the way in which business process definitions say they ought to work
  - Requirements derived from cooperation and awareness of other people’s activities
    - Awareness of what other people are doing leads to changes in the ways in which we do things.
  - Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

Ethnography and prototyping for requirements analysis

- 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.



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.
- Scenario is 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.

View Points

- Viewpoint-oriented approaches to requirements engineering organise both the elicitation process and the requirements themselves using different viewpoints.
- A viewpoint is a way of organising the requirements for a software system, based on some perspective such as an end-user perspective or a manager's perspective.
- A key strength of viewpoint-oriented analysis is that it recognises the existence of multiple perspectives and provides a framework for discovering conflicts in the requirements proposed by different stakeholders.
- Viewpoints can be used as a way of classifying different types of stakeholder and other sources of requirements.
- There are three generic types of viewpoint:

1. **Interactor viewpoints** that represent people or other systems that interact directly with the system. In the bank ATM system, examples of interactor viewpoints are the bank's customers and the bank's account database.
2. **Indirect viewpoints** that represent stakeholders who do not use the system themselves but who influence the requirements in some way. In the bank ATM system, examples of indirect viewpoints are the management of the bank and the bank security staff.
3. **Domain viewpoints** that represent domain characteristics and constraints that influence the system requirements. In the bank ATM system, an example of a domain viewpoint would be the standards that have been developed for inter-bank communications.

Typically, these viewpoints will provide different types of requirement.

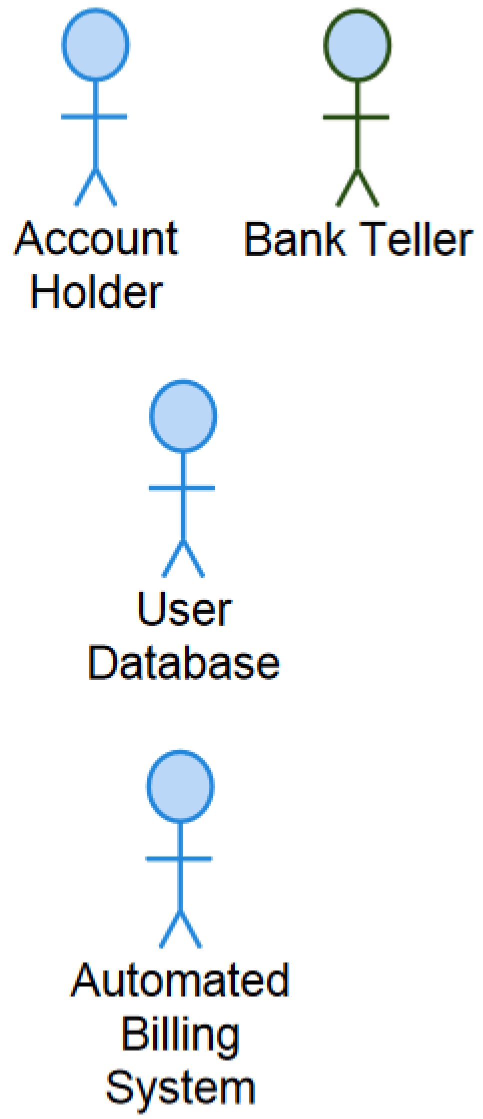
- Interactor viewpoints provide detailed system requirements covering the system features and interfaces.
- Indirect viewpoints are more likely to provide higher-level organisational requirements and constraints.
- Domain viewpoints normally provide domain constraints that apply to the system.

Use Cases:

- A use case approach defines what a system will do from a user's perspective.
- They provide a detailed description of a particular user requirement and are structured.
- Description of a sequence of interactions between a system and external actors
- Developed by Ivar Jacobson
  - Not exclusively for object-oriented analysis
- Actors – any agent that interact with the system to achieve a useful goal (e.g., people, other software systems, hardware)
- Use case describes a typical sequence of actions that an actor performs in order to complete a given task
- The objective of use case analysis is to model the system
  - From the point of view of how actors interact with this system
  - when trying to achieve their objectives
- A use case model consists of
  - A set of use cases
  - An optional description or diagram indicating how they are related
- A use case should describe the user's interaction with the system ...
  - Not the computations the system performs
- In general, a use case should cover the full sequence of steps from the beginning of a task until the end

Use case

- This technique combines text and pictures to provide a better understanding of the requirements.
- The use cases describe the 'what', of a system and not 'how'. So it covers the functional requirements.
- Hence, they only give a functional view of the system. The components of the use case design includes three major things – Actor, Use cases, use case diagram.
- **1.Actor:**
    - It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.
        - Primary actors – It requires assistance from the system to achieve a goal.
        - Secondary actor – It is an actor from which the system needs assistance.
- **2.Use cases :**
    - They describe the sequence of interactions between actors and the system.
    - They capture who(actors) do what(interaction) with the system.
    - A complete set of use cases specifies all possible ways to use the system.
- **3.A use case diagram :**
    - A use case diagram graphically represents what happens when an actor interacts with a system.
    - It captures the functional aspect of the system.
        - A stick figure is used to represent an actor.
        - An oval is used to represent a use case.
        - A line is used to represent a relationship between an actor and a use case.



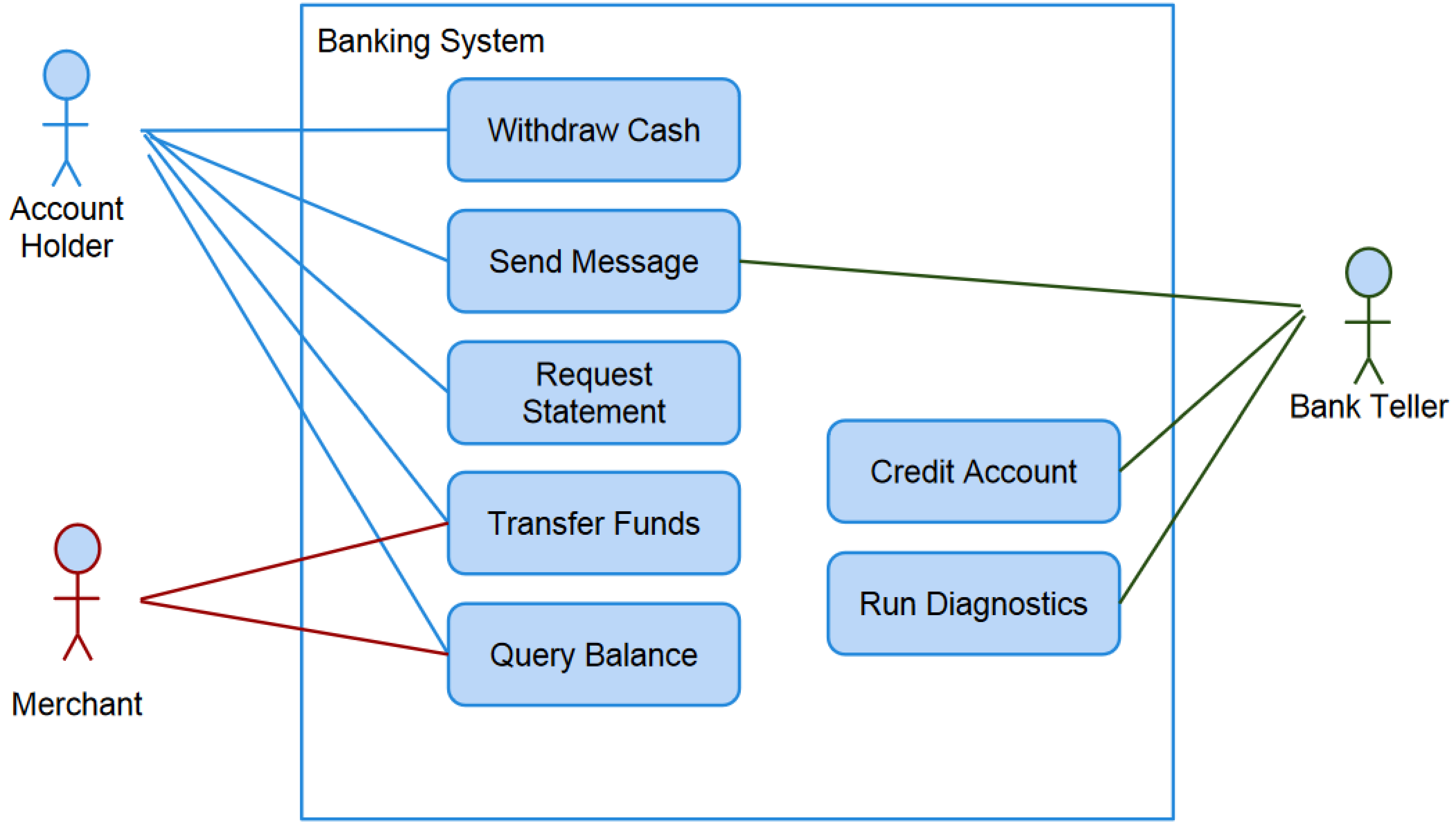
An **actor** is a role a user plays with respect to the system.

- Actors carry out use cases. An actor can perform many use cases. A use case can involve multiple actors.
- A single user can be multiple actors, depending on how they use a system.
- Actors do not need to be human - can be an external system (hardware or software) that interacts with the system being built.

Use cases:cash withdrawl

- **Cash Withdrawal: Correct PIN**
- The customer inserts the card into the ATM. The ATM accepts the card and asks the user for the PIN. If the PIN is correct, the ATM asks the user for an account choice. The user enters the account. The ATM asks the user for a cash amount. The user enters the amount. The ATM asks the user to verify the account. The user verifies the account. If there are sufficient funds in the account, the money is dispensed and the amount is withdrawn from the account.
- **Scenarios and use case description**
  - A Scenario is one possible sequence of user interactions.
  - A use case description encompasses all of the scenarios that can occur when attempting to achieve a particular user goal.

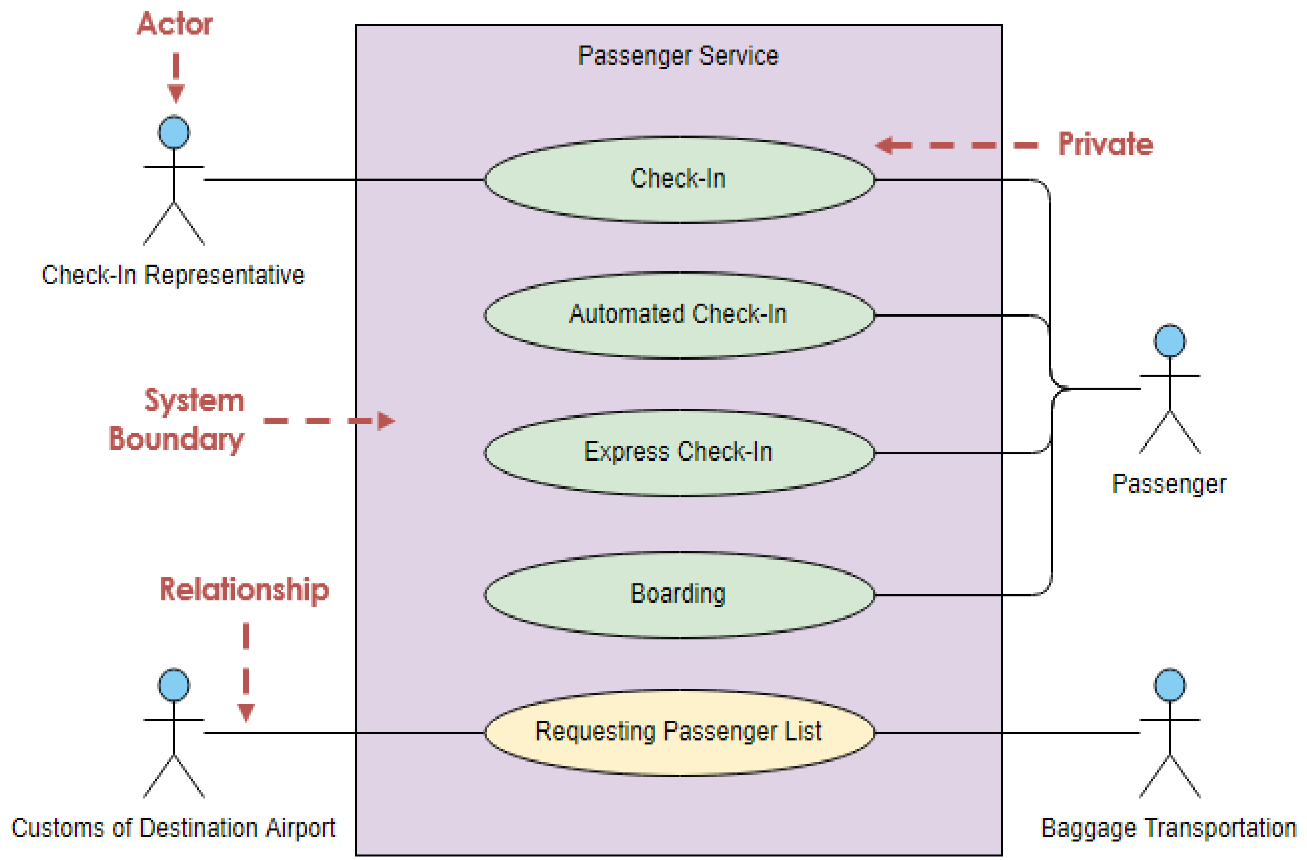
Use Case Diagram:



Use Case Modeling

- Use Case is the Depiction of a system's behavior or functionality under various conditions as the system responds to requests from users
- A use case describes how a user uses a system to accomplish a particular goal.
- A use case diagram consists of the system, the related use cases and actors and relates these to each other to visualize: what is being described? (**system**), who is using the system? (**actors**) and what do the actors want to achieve? (**use cases**), thus, use cases help ensure that the correct system is developed by capturing the requirements from the user's point of view.
- A use-case diagram can help provide a higher-level view of the system, providing the simplified and graphical representation of what the system must actually do.
- A use case (or set of use cases) has these **characteristics**:
  - Organizes functional requirements
  - Models the goals of system/actor (user) interactions
  - Describes one main flow of events (main scenarios) and possibly other exceptional flows (alternatives), also called paths or user scenarios
- **Purpose of Use Case Diagram**
  - Specify the context of a system
  - Capture the requirements of a system
  - Validate a systems architecture
  - Drive implementation and generate test cases
  - Developed by analysts together with domain experts

- A use case diagram notations
  - Use cases
  - Actor
  - Use Case relationships
  - System Boundary



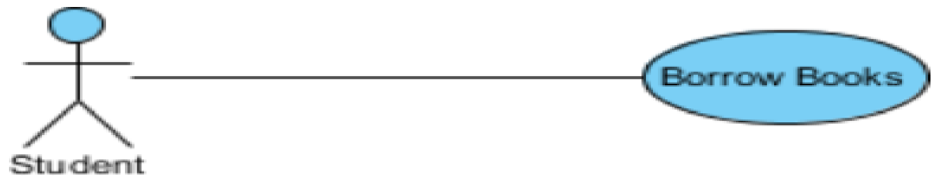
Use Case Components

- **Actor**
  - Actor is an external entity that interacts with the system.
  - Most actors represent user roles, but actors can also be external systems.
  - An actor is a role, not a specific user; one user may play many roles, and an actor may represent many users.
  - An Actor is outside or external the system.
  - It can be a:
    - Human
    - Peripheral device (hardware)
    - External system or subsystem
    - Time or time-based event
  - Represented by stick figure

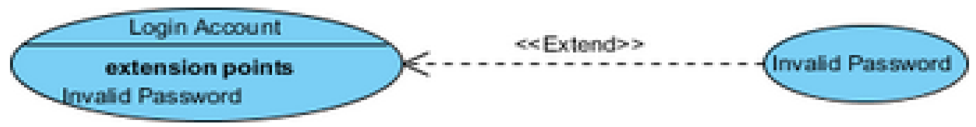
- **Boundary**
  - A boundary is the dividing line between the system and its environment.
  - Use cases are within the boundary.
  - Actors are outside of the boundary.

- **Cases:**
  - A use case describes how actors uses a system to accomplish a particular goal
  - . Use cases are typically initiated by a user to fulfill goals describing the activities and variants involved in attaining the goal.
  - Document containing detailed specifications for a use case
  - Contents can be written as simple text or in a specified format

- Relationships
  - **Association**
    - Associations are used to describe the relationships between actors and the use cases they participate in.
    - This relationship is commonly known as a "communicates-association".
    - Represented by solid line



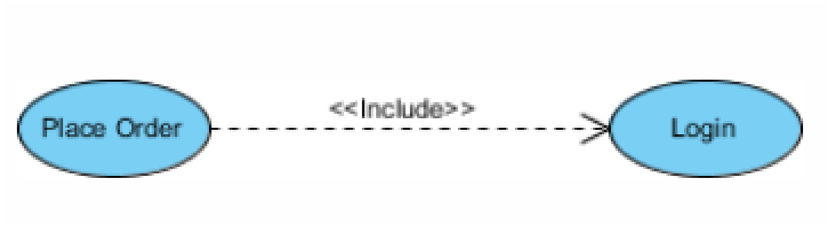
  - **Extends**
    - <<extend>> is used to include optional behavior from an extending use case in an extended use case.



    - Indicate that an **"Invalid Password"** use case may include (subject to specified in the extension) the behavior specified by base use case **"Login Account"**.
    - Depict with a directed arrow having a dotted line. The tip of arrowhead points to the base use case and the child use case is connected at the base of the arrow.
    - The stereotype "<<extends>>" identifies as an extend relationship

Use Case Relationships

- **Includes**
  - <<include>> is used to include common behavior from an included use case into a base use case in order to support re-use of common behavior.
  - When a use case is depicted as using the functionality of another use case, the relationship between the use cases is named as include or uses relationship.
  - A use case includes the functionality described in another use case as a part of its business process flow.
  - An include relationship is depicted with a directed arrow having a dotted line. The tip of arrowhead points to the child use case and the parent use case connected at the base of the arrow.
  - The stereotype "<<include>>" identifies the relationship as an include relationship.



- **Inheritance(Generalization/Specialization)**
  - A generalization relationship is a parent-child relationship between use cases.
  - The child use case is an enhancement of the parent use case.
  - Generalization is shown as a directed arrow with a triangle arrowhead.
  - The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case.
  - A generalization relationship means that a child use case inherits the behavior and meaning of the parent use case.
  - The child may add or override the behavior of the parent.

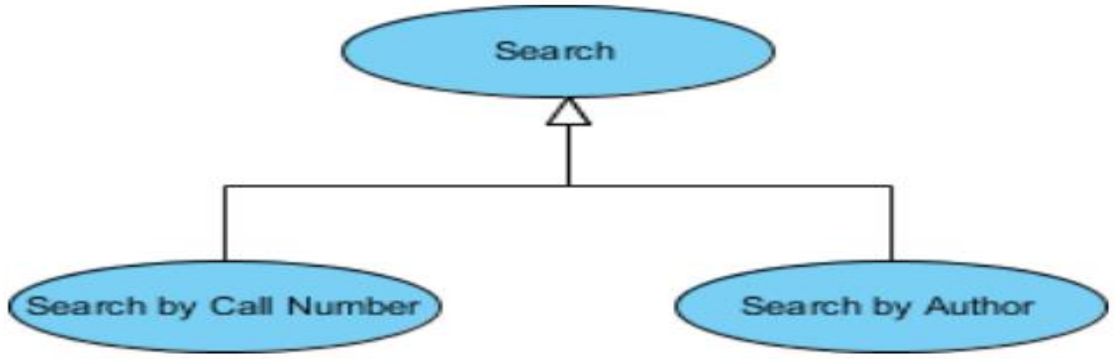
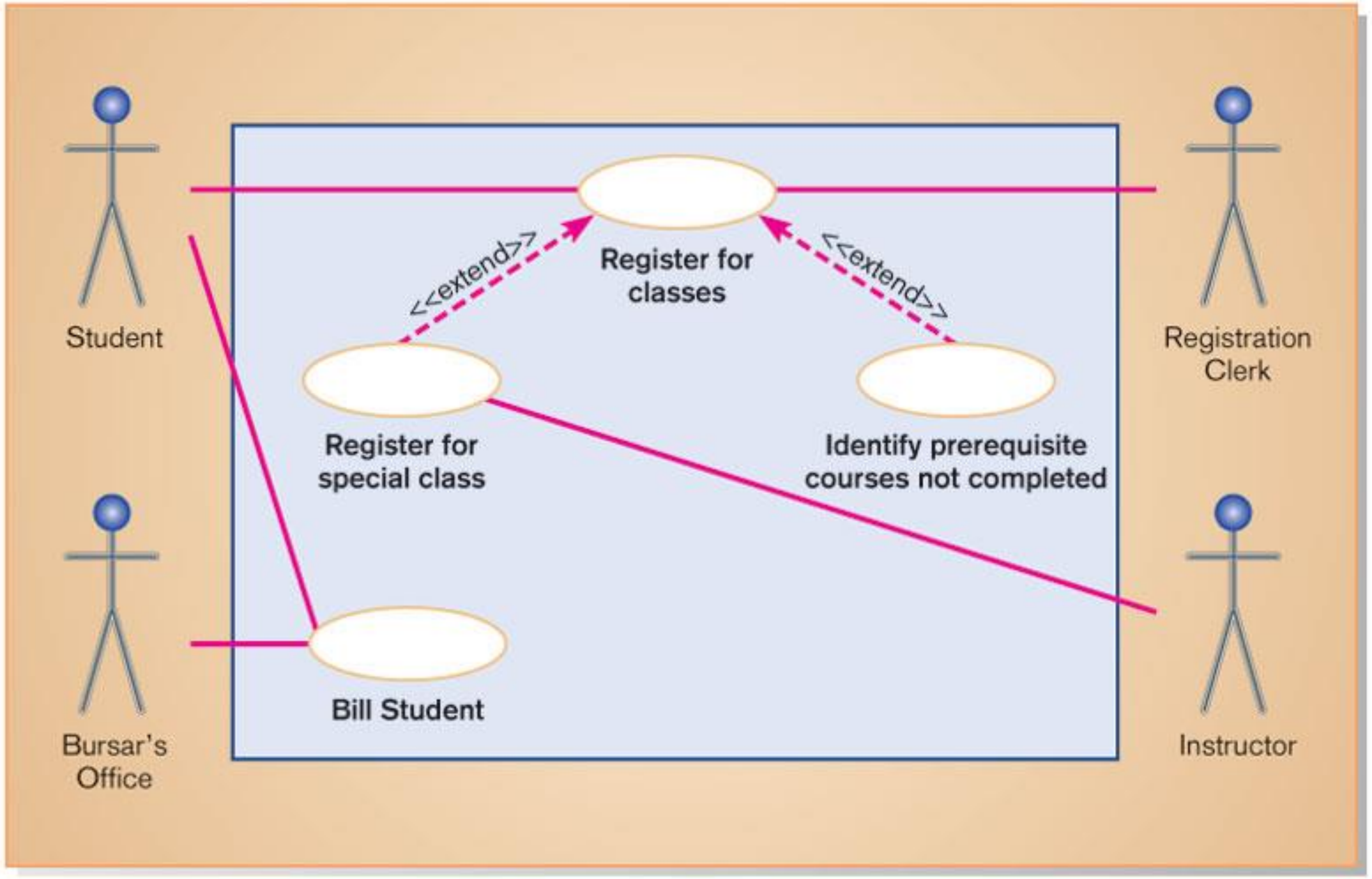


Figure 7-22 A use case diagram for a university registration system

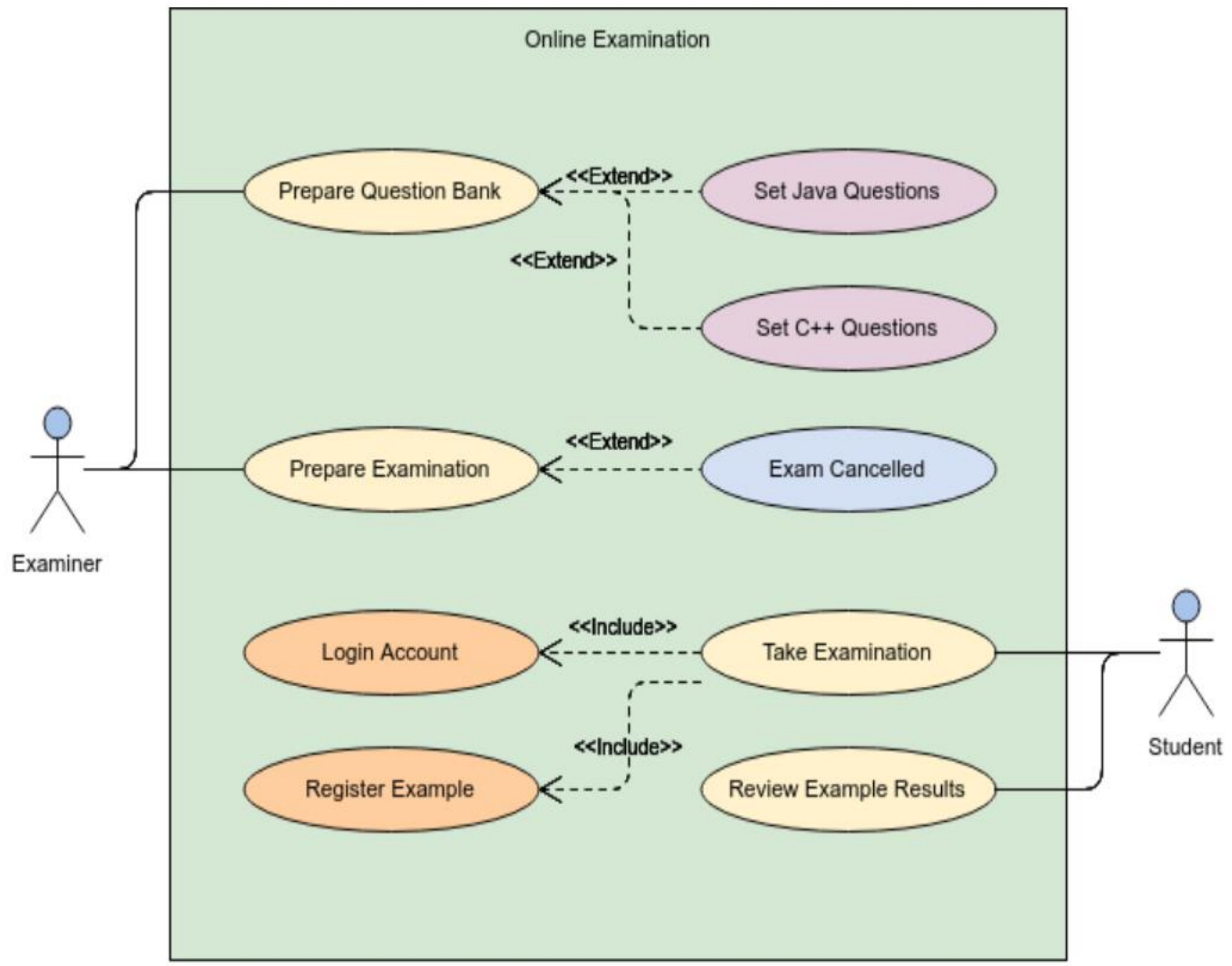


Guidelines to draw Use Case Diagram

- Identify the Actors (role of users) of the system.
- For each category of users, identify all roles played by the users relevant to the system. Identify the cases exists in the system.
- Create use cases for every goal and assign the different types of relationships as per needed.
- Structure the use cases.i.e organize different actors, cases and cases relationships.
- Prioritize, review, estimate and validate the users.

Advantages of Use case diagram

- Use cases is a powerful technique for the elicitation and documentation of functional requirements.
- Because, use cases are easy to understand and provide an excellent way for communicating with customers and users as they are written in natural language.
- Use cases can help manage the complexity of large projects by partitioning the problem into major user features (i.e., use cases) and by specifying applications from the users' perspective.
- A use case scenario, often represented by a sequence diagram, involves the collaboration of multiple objects and classes, use cases help identify the messages (operations and the information or data required — parameters) that glue the objects and classes together.
- Use cases provide a good basis to link between the verification of the higher-level models (i.e. interaction between actors and a set of collaborative objects), and subsequently, for the validation of the functional requirements (i.e. blueprint of white-box test).
- Use case driven approach provides an traceable links for project tracking in which the key development activities such as the use cases implemented, tested, and delivered fulfilling the goals and objectives from the user point of views.



Requirement Specification

- Requirement Specification is the process of writing down the **user and system requirements** in a requirements document.
- **User requirements** have to be understandable by end-users and customers who do not have a technical background.
- **System requirements** are more detailed requirements and may include more technical information.
- **System requirements** are expanded versions of the user requirements that software engineers use as the starting point for the system design.
- **System requirements** add detail and explain how the system should provide the user requirements.
- **System requirements** may be used as part of the contract for the implementation of the system and should therefore be a complete and detailed specification of the whole system.
- Ideally, **the user and system requirements** should be clear, unambiguous, easy to understand, complete, and consistent.
- In practice, this is almost impossible to achieve.
- Stakeholders interpret the requirements in different ways, and there are often inherent conflicts and inconsistencies in the requirements.
- **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.**
- The user requirements for a system should describe the functional and nonfunctional requirements so that they are understandable by system users who don't have detailed technical knowledge.
- Ideally, they should specify only the external behavior of the system.
- The requirements document should not include details of the system architecture or design.
- Consequently, if you are writing user requirements, you should not use software jargon, structured notations, or formal notations.
- We should write user requirements in natural language, with simple tables, forms, and intuitive diagrams.

Ways of writing a system requirements specification

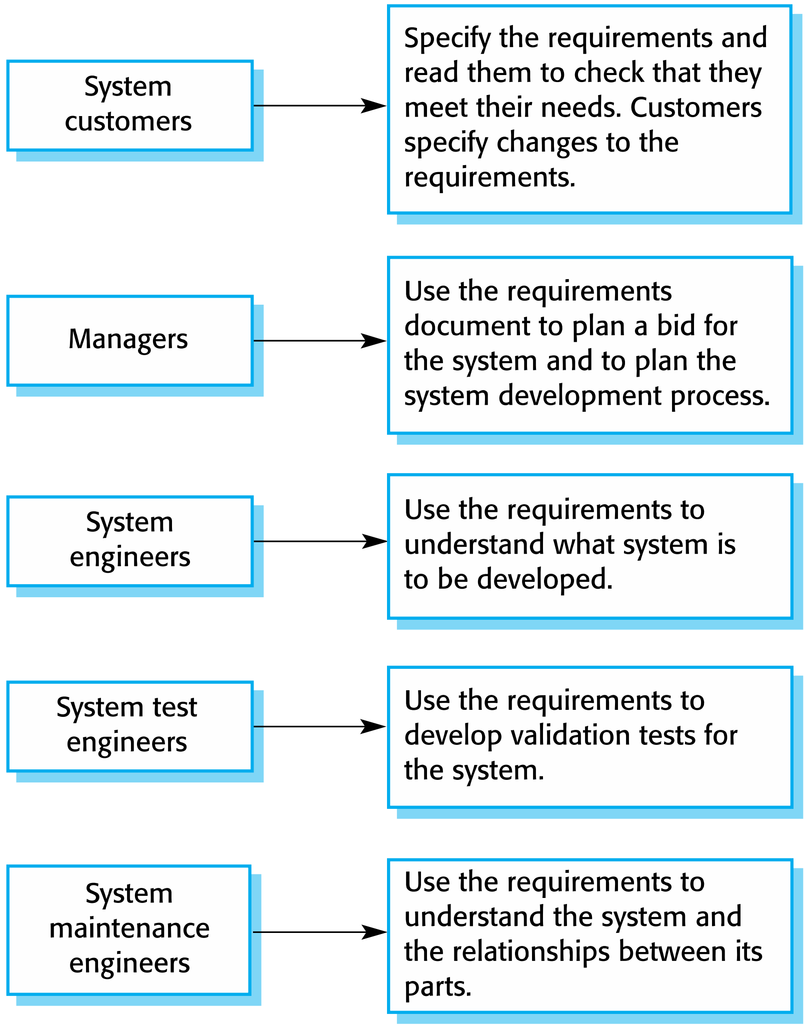
Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML 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 are reluctant to accept it as a system contract

- **Natural Language Specifications:**
  - Requirements are written as natural language sentences supplemented by diagrams and tables.
  - Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.
    - **Guidelines for writing Natural Language Specifications**
      - Invent a standard format and use it for all requirements.
      - Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.
      - Use text highlighting to identify key parts of the requirement.
      - Avoid the use of computer jargon.
      - Include an explanation (rationale) of why a requirement is necessary.
    - **Problems with natural language**
      - Lack of clarity
        - Precision is difficult without making the document difficult to read.
      - Requirements confusion
        - Functional and non-functional requirements tend to be mixed-up.
      - Requirements amalgamation
        - Several different requirements may be expressed together.

- **Structured specifications**
  - An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.
  - This works well for some types of requirements e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.
- **Form-based specifications**
  - Definition of the function or entity.
  - Description of inputs and where they come from.
  - Description of outputs and where they go to.
  - Information about the information needed for the computation and other entities used.
  - Description of the action to be taken.
  - Pre and post conditions (if appropriate).
  - The side effects (if any) of the function.
- **Tabular Specifications**
  - Used to supplement natural language.
  - Particularly useful when you have to define a number of possible alternative courses of action.
  - For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.

Software Requirement Document

- 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. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.
- Information in requirements document depends on type of system and the approach to development used.
- Systems developed incrementally will, typically, have less detail in the requirements document.
- Requirements documents standards have been designed e.g. IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.


System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System engineers
Use the requirements to understand what system is to be developed.
System test engineers
Use the requirements to develop validation tests for the system.
System maintenance engineers
Use the requirements to understand the system and the relationships between its parts.

Users of a requirements document

Structured of the Requirement Document

- Preface
  - Define the expected readership of the document
  - Describe its version history
  - A rational for the creation of a new version
  - A summary of the changes made in each version
- Introduction
  - Describe the need for the system
  - Describe the system's functions
  - Explain how it will work with other systems
  - Describe how the system fits into the overall business or strategic objectives of the organization commissioning the software
- Glossary
  - Define the technical terms used in the document
  - Should not make assumptions about the experience or expertise of the reader
- User requirements definition
  - Describe the services provided for the user
  - Non-functional system requirements should also be described
  - May use natural language, diagrams or other notations that are understandable to customers
  - Product and process standards should be specified (if applicable)
- System architecture
  - A high-level overview of the anticipated system architecture
  - Showing the distribution of functions across system modules
  - Architectural components should be highlighted (that are reused)
- System requirements specification
  - Describe the functional and non-functional requirements in more detail
  - Optional: further detail to the non-functional requirements
  - Optional: interfaces to other systems
- System models
  - Include graphical system models showing the relationships between the system components, the system and its environment
  - Examples: object models, data-flow models, semantic data models
- System evolution
  - Describe the fundamental assumptions on which the system is based
  - Any anticipated changes due to hardware evolution, changing user needs etc.
  - Useful for system designers: may help to avoid design decisions
- Appendices
  - Provide detailed, specific information that is related to the application being developed
  - Example: hardware and database descriptions
  - Hardware requirements: minimal and optional configurations for the system
  - Database requirements: logical organization of the data used by the system and the relationships between data
- Index
  - A normal alphabetic index
  - An index of diagrams
  - An index of functions etc.

Software Requirement Specification Document(SRS)

- An SRS is a Software Requirement Specification document which serves as a written contract between client and an organization.
- A Software requirements specification document describes the intended purpose, requirements and nature of a software to be developed. It also includes the yield and cost of the software.
- A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fulfill all stakeholders (business, users) needs.

Why SRS?

- A software requirements specification is the basis for your entire project. It lays the framework that every team involved in development will follow.
- It’s used to provide critical information to multiple teams — development, quality assurance, operations, and maintenance. This keeps everyone on the same page.
- Using the SRS helps to ensure requirements are fulfilled. And it can also help you make decisions about your product’s lifecycle — for instance, when to retire a feature.
- Writing an SRS can also minimize overall development time and costs. Embedded development teams especially benefit from using an SRS
- It acts as a legal contract document between the customer and the developer.

- Characteristics/Properties of Good SRS
  - Concise
  - Structured
  - Integrate
  - Verifiable
  - Must hold following characteristics
    - Correctness
    - Completeness
    - Unambiguous
    - Modifiable
    - Traceable
    - Consistent
    - Feasible etc

What does the SRS document need to address?

- Functionality
  - What is the software supposed to do?
- External Interfaces
  - How does the software interact with people, the system's hardware, other hardware, and other software?
- Performance
  - What is the speed, availability, response time, recovery time, etc of the software functions?
- Attributes
  - What are the considerations for portability, maintainability, security, etc?
- Design/Implementation Constraints
  - Are there any required standards in effect, implementation language, policies for data management, resource limits, operating environment

Requirement Validation

- This activity checks the requirements for realism, consistency and completeness.
- In this process, errors in the requirements are discovered.
- It must then be modified to correct these problems.
- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
    - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.
- It includes checks like: *validity check, consistency check, completeness check, realism check, verifiability check*
- Requirement validation techniques
    - **Requirement reviews:** the requirements are analyzed systematically by a team of reviewers.
        - Requirement reviews can be of informal type and formal type.
        - **Informal reviews:**
            - simply involve contractors discussing requirements with as many system stakeholders as possible.
        - **formal review:**
            - the development team should 'walk' the client through the system requirements explaining the implications of each requirement.
    - **Prototyping:** an executable model of the system is demonstrated to end-users and customers. They can experiment with this model to see if it meets their real needs.
    - **Test-case generation:** If a test is difficult or impossible to design, this usually means that requirement will be difficult to implement and should be reconsidered.

Requirement Management

- Requirement management is the process of managing changing requirements during the requirement 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 view points have different requirements and these are often contradictory.
- New requirements emerge as a system is being developed and after it has gone into use.
- We need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes.
- We 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.