CHAPTER FOUR

Requirement elicitation

AN OVERVIEW OF REQUIREMENTS ELICITATIONS

- IEEE defines a requirement as
  - A condition of capability needed by a user to solve a problem or achieve an objective;
  - A condition or a capability that must be met or possessed by a system ... to satisfy a contract, standard, specification, or other formally imposed document."

- Problem of scale is a key issue for SE
- For small scale, understand and specifying requirements is easy.
- As systems grew more complex, it became evident that the goals of the entire system could not be easily comprehended.
  - Hence we need more rigorous requirements analysis

- In the beginning, these needs are in the minds of various people in the client organization.
- The requirements analyst has to identify the requirements by talking to these people and understanding their needs.

- In system were there is existing manual system requirements can be found by observing the current practice.
- But no such methods exist for systems for which manual processes do not exist or for "new features"

- For such system needs and requirements of the system many not be known even to the user—they have to be visualized and created.
- The requirements phase translates the ideas in the minds of the clients (the input), into a formal document (the output of the requirements phase).
    - Requirement specification and analysis phase ends with a software requirements specification (SRS) document
    - SRS specifies what the proposed system should do without describing how the software will do it.

- Requirements understanding is hard because
  - Visualizing a future system is difficult
  - Capability of the future system not clear, hence needs not clear
  - Requirements change with time
- Essential to do a proper analysis and specification of requirements

NEED FOR SRS

- SRS establishes basis of agreement between the user and the supplier.
  - Users needs have to be satisfied, but user may not understand software
  - Developers will develop the system, but may not know about problem domain
  - SRS is the medium to bridge the communication gap and specify user needs in a manner both can understand

- Helps user understand his needs.
  - users do not always know their needs
  - the goal is not just to automate a manual system, but also to add value through IT
  - The requirement process helps clarify needs

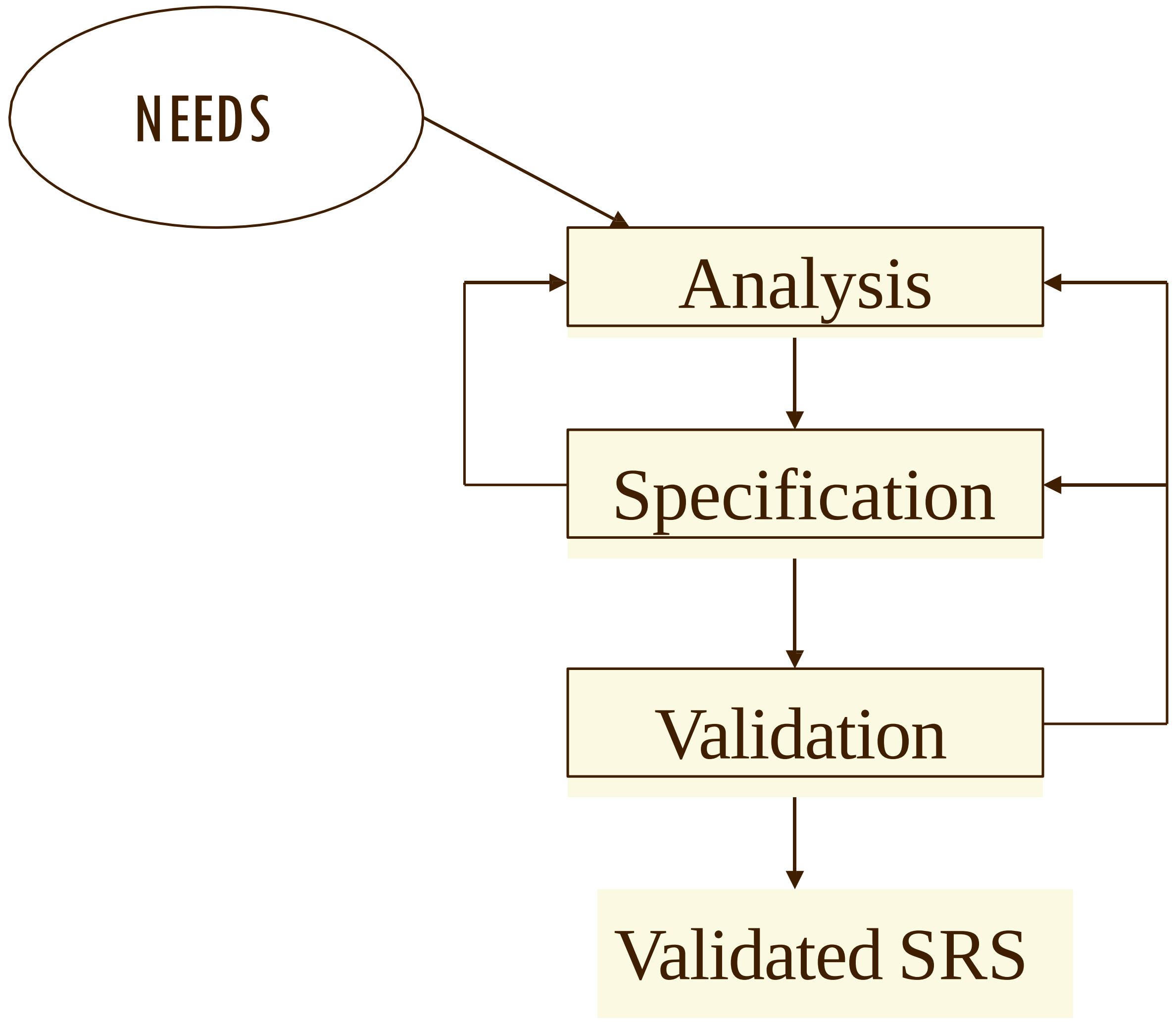
- SRS provides a reference for validation of the final product
  - helps the client determine if the software meets the requirements.
  - Without a proper SRS, there is no way a client can determine if the software being delivered is what was ordered, and there is no way the developer can convince the client that all the requirements have been fulfilled.

- High quality SRS essential for high Quality SW
  - Most errors discovered during testing are originated from requirement and early designing stages.
  - Requirement errors get manifested in final software
  - to satisfy the quality objective, must begin with high quality SRS

- Good SRS reduces the development cost
  - SRS errors are expensive to fix later
  - Requirement changes can cost a lot
  - Good SRS can minimize changes and errors
  - Substantial savings; extra effort spent during req. saves multiple times that effort

REQUIREMENT PROCESS

- Sequence of steps that need to be performed to convert user needs into SRS
- Process has to elicit needs and requirements and clearly specifies it
- Basic activities
  - problem or requirement analysis
  - requirement specification
  - validation


NEEDS
Analysis
Specification
Validation
Validated SRS

- Requirement analysis (hardest): is to obtain a thorough understanding of what the software needs to provide.
- Requirement specification : specifying the requirements in a document.
- Requirements validation: focuses on ensuring that what has been specified in the SRS are indeed all the requirements of the software and making sure that the SRS is of good quality.

- Requirement process is not linear, it is iterative and parallel
- Specification itself may help analysis
- Validation can show gaps that can lead to further analysis and spec

- Focus of analysis is on understanding the desired systems and it’s requirements
- Divide and conquer is the basic strategy for complex systems.
    - decompose into small parts, understand each part and relation between parts

- Large volumes of information is generated
  - organizing them is a key
- Techniques like data flow diagrams, object diagrams etc. used in the analysis

PROBLEM ANALYSIS

- Aim: to gain an understanding of the needs, requirements, and constraints on the software
- Analysis involves
  - interviewing client and users
  - reading manuals (Document analysis)
  - studying current systems (Observation)
  - helping client/users understand new possibilities
  - Brainstorming
  - Focus Group
  - Prototyping
  - Survey/questionnaire

- The informal approach to analysis is one where no defined methodology is used.
- with this approach no formal model is built of the system.
- The problem and the system model are essentially built in the minds of the analysts

- the analyst will have a series of meetings with the clients and end users.

CHARACTERISTICS OF AN SRS

- **Correct:** if every requirement included in the SRS represents something required in the final system.
- **Complete:** if everything the software is supposed to do and the responses of the software to all classes of input data are specified in the SRS.

- **Unambiguous:** if and only if every requirement stated has one and only one interpretation.
- **Verifiable:** if and only if every stated requirement is verifiable.
- **Consistent:** if there is no requirement that conflicts with another.

- **Modifiable:** if its structure and style are such that any necessary change can be made easily while preserving completeness and consistency.
- **Traceable:** if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development.

- ***Forward traceability:*** means that each requirement should be traceable to some design and code elements.
- ***Backward traceability:*** requires that it be possible to trace design and code elements to the requirements they support.

TYPES OF SOFTWARE SYSTEM REQUIREMENTS

- There are two types of software system requirement,
    - Functional Requirements
    - Non functional requirements

FUNCTIONAL REQUIREMENT

- These are statements of services the system should provide,
  - how the system should react to particular inputs, and
  - how the system should behave in particular situations.

EXAMPLE

- A user shall be able to search the appointments lists for all clinics.
- The system shall generate each day, 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 eight-digit employee number.
- ……………..

NONFUNCTIONAL REQUIREMENTS

- These are requirements that are not directly concerned with the specific services delivered by the system to its users.
- Nonfunctional requirements can be elicited by investigating the following issues.

- **User interface and human factors:** What kind of interface should the system provide? What is the level of expertise of the users?
- **Documentation:** What level of document is required? Should only user documentation be provided? Should there be technical documentation for maintainers? Should the development process be documented?

- **Hardware considerations:** Are there hardware compatibility requirements? Will the system interact with other hardware systems?
- **Performance characteristics:** How responsive should the system be? How many concurrent users should it support? What is a typical or extreme load?

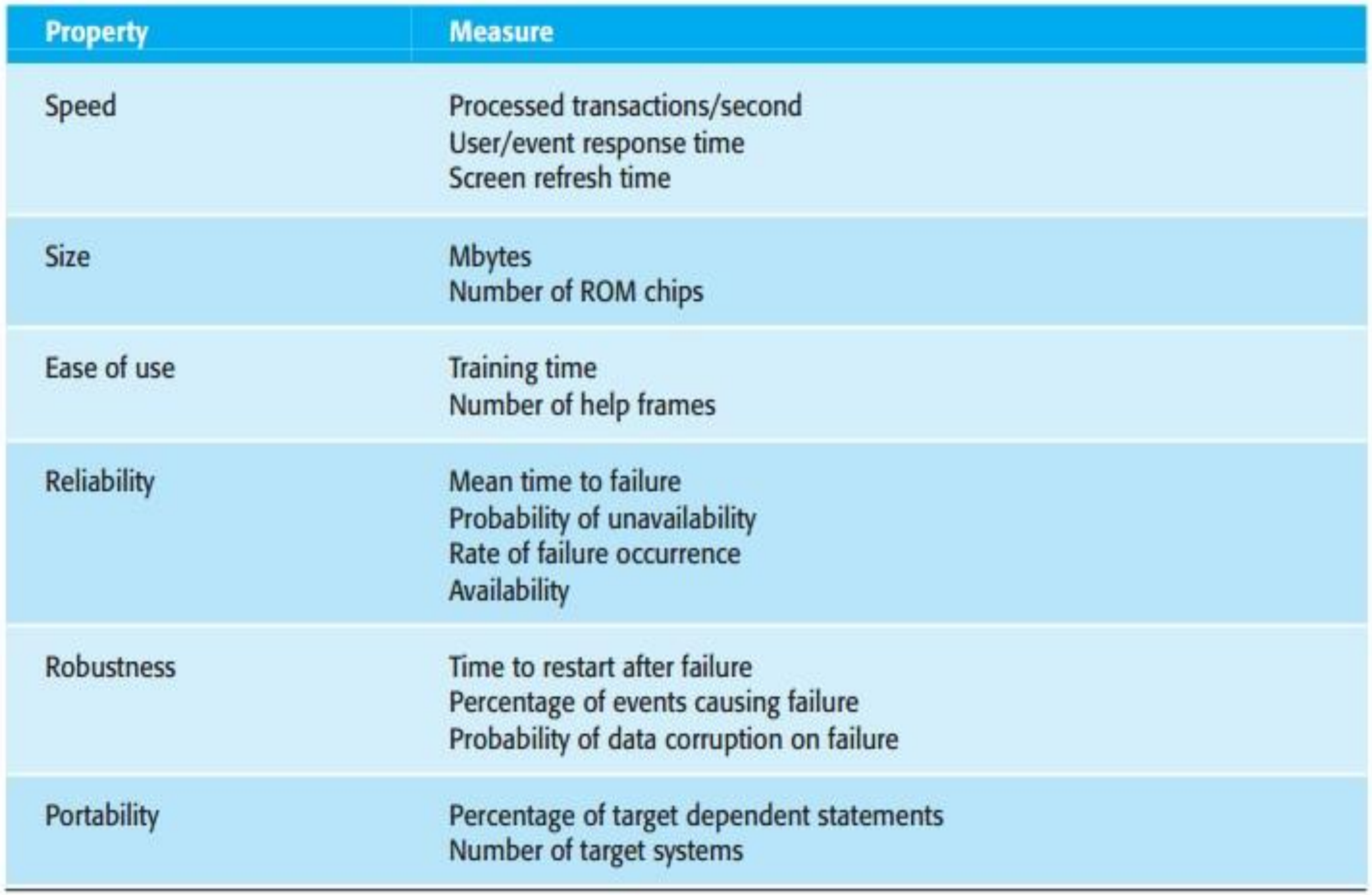
- **Error handling and extreme conditions:** How should the system handle exceptions? Which exceptions should the system handle? What is the worse environment in which the system is expected to perform? Are there safety requirements on the system?

- **Quality issues:** How reliable/ available/ robust should the system be ? What is the client’s involvement in assessing the quality of the system or the development process?

- **System modifications:** What is the anticipated scope of future changes? Who will perform the changes?
- **Physical environment:** Where will the system be deployed? Are there external factors such as weather conditions that the system should withstand?

- **Security issues:** Should the system be protected against external intrusions or against malicious users? To what level?
- **Resource issues:** What are the constraints on the resources consumed by the system?

SOME NONFUNCTIONAL REQUIREMENT WITH THEIR MEASUREMENTS



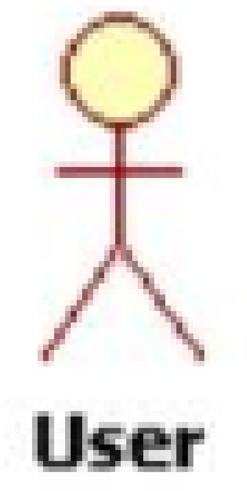
Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart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

REQUIREMENTS ELICITATION ACTIVITIES

- A use case describes how a user uses a system to accomplish a particular goal.
- A use case diagram consists of
  - the system (what is being described?),
  - The actors (who is using the system?) and
  - the use cases (what do the actors want to achieve?)
- In brief, the purposes of use case diagrams can be said to be as follows –
  - Used to gather the requirements of a system.
  - Used to get an outside view of a system.
  - Identify the external and internal factors influencing the system.
  - Show the interaction among the requirements are actors.

IDENTIFYING ACTORS

- The operating environment of a software system consists of the users, devices, and programs that the system interacts with. These are called **actors**.
- A UML actor icon is a stick figure:

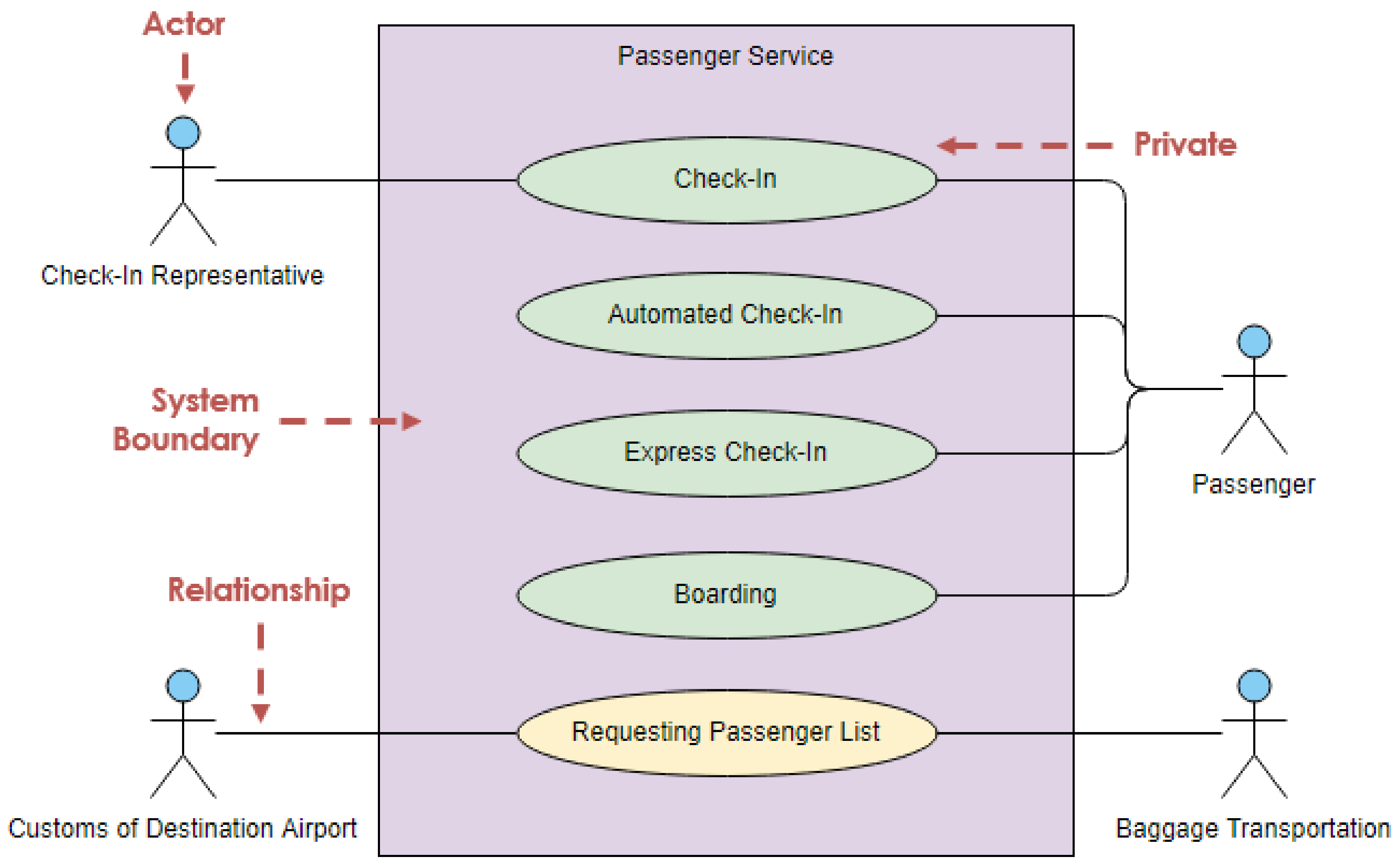


- Types of actors:
  - primary - a user whose goals are fulfilled by the system
  - supporting - provides a service (e.g., info) to the system
  - offstage - has an interest in the behavior but is not primary or supporting, e.g., government • importance: ensure all interests (even subtle) are identified and satisfied

HEURISTICS OF FINDING ACTORS

- Which user groups require help from the system to perform their tasks?
- Which user groups are needed to execute the system‘s most obvious main functions?
- Which user groups are required to perform secondary functions, such as maintenance and administration?
- Will the system interact with any external hardware or software system?

USE CASE DIAGRAM NOTATIONS

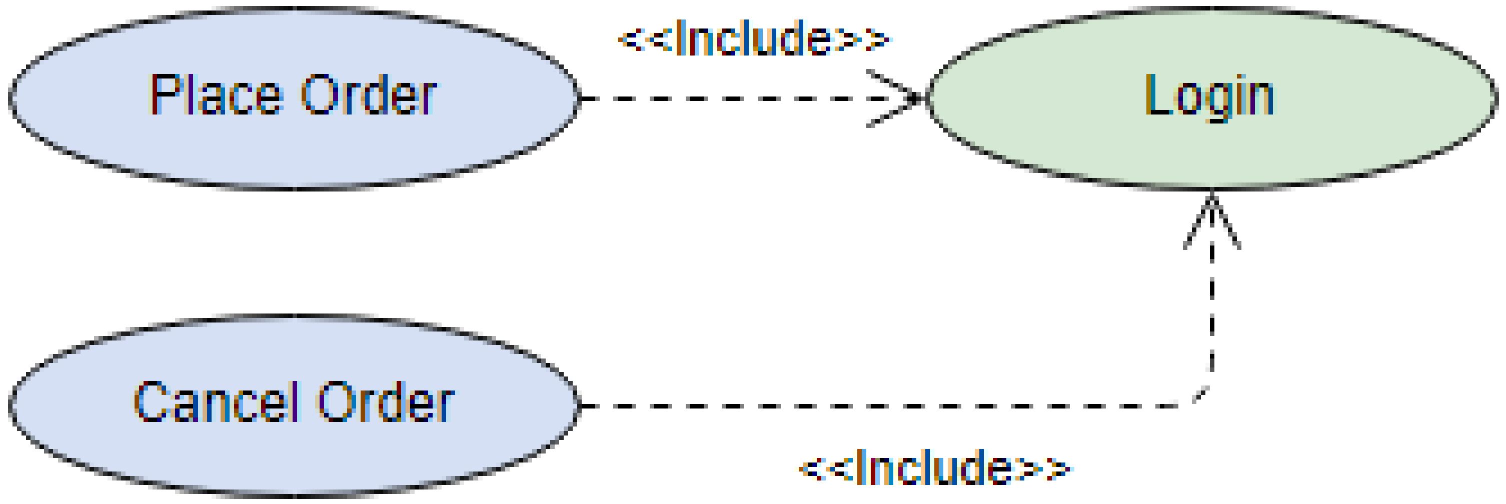


DESCRIPTION OF USE CASE COMPONENTS

- **Actor**
  - Actors are usually individuals involved with the system defined according to their roles. The actor can be a human or other external system.
- **Use Case**
  - 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.
- **Relationship**
  - The relationships between and among the actors and the use cases.
- **System Boundary**
  - The system boundary defines the system of interest in relation to the world around it.

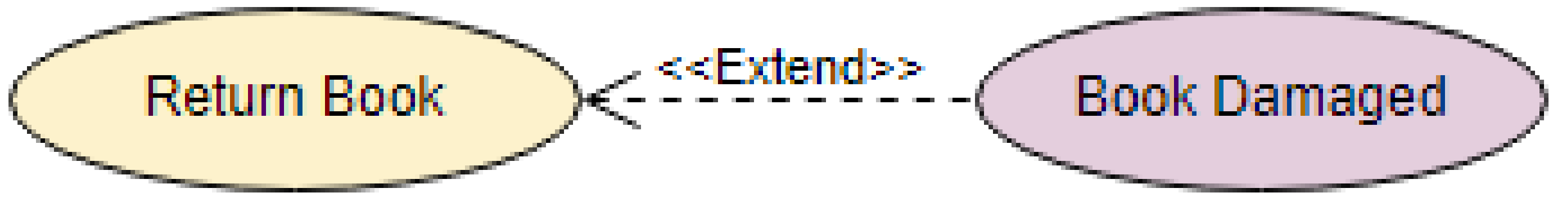
STRUCTURING USE CASES

- UML defines three stereotypes of association between Use Cases:
- <<include>> Use Case
  - The time to use the <<include>> relationship is after you have completed the first cut description of all your main Use Cases.
  - You can now look at the Use Cases and identify common sequences of user-system interaction.



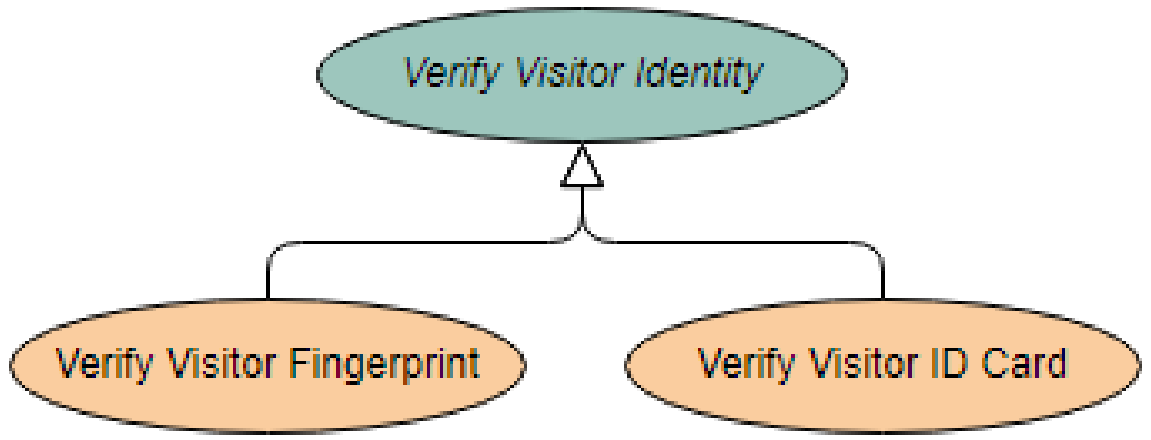
❑<<extend>> Use Case

- An extending use case is, effectively, an alternate course of the base use case.
- The <<extend>> use case accomplishes this by conceptually inserting additional action sequences into the base use-case sequence.

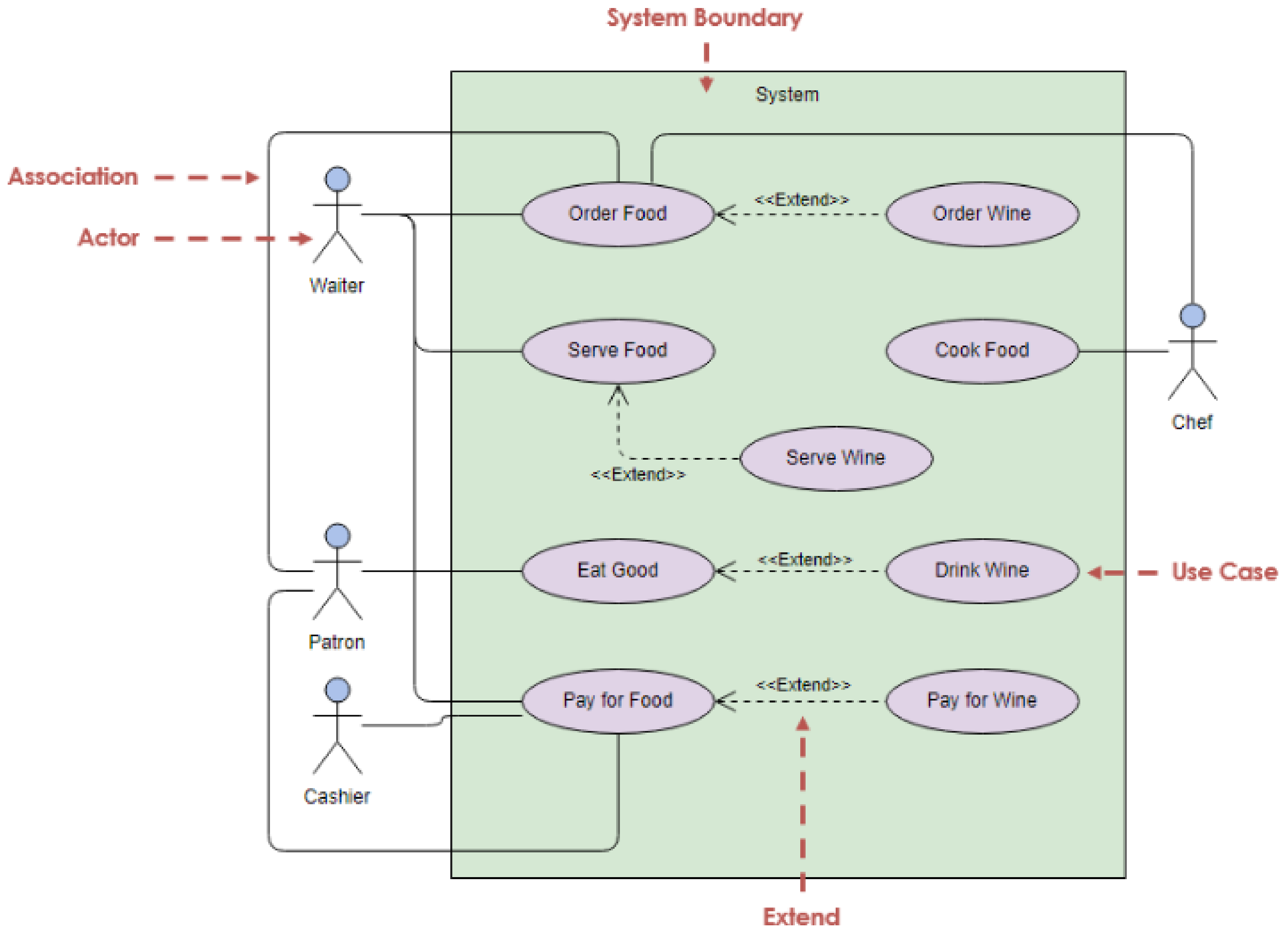


Abstract and generalized Use Case

- The general use case is abstract. It can not be instantiated, as it contains incomplete information. The title of an abstract use case is shown in italics.



Example: restaurant (the business system) and its primary actors.



- Exercise: online shopping system.

TEXTUAL DESCRIPTION OF USE CASES

- It is helpful to have our use cases textual description.
- For the textual description of a use case, we use a template composed of six fields
  - **Name:**
    - ✓The name of the use case is unique across the system so that developers (and project participants) can unambiguously refer to the use case.
  - **Participating actors:**
    - ✓Participating actors are actors interacting with the use case.
  - **Entry conditions:**
    - ✓Entry conditions describe the conditions that need to be satisfied before the use case is initiated.

- **Flow of events :**
  - ✓The flow of events describes the sequence of interactions of the use case, which are to be numbered for reference.
- **Exit conditions:**
  - ✓Exit conditions describe the conditions satisfied after the completion of the use case.
- **Quality requirements:**
  - ✓Quality requirements are requirements that are not related to the functionality of the system. These include constraints on the performance of the system, its implementation, the hardware platforms it runs on, and so on.

Use Case Name: AddUserAccount

Participating Actor: Administrator

Description: Allow the administrator to add accounts.

Entry conditions: Administrator should log in.
Flow of Events:

1. The administrator initiates "Add User Accounts" use case.
2. The System displays the Create User form.
3. The administrator fills in the details of the new user like name, Id, role, user name, and password.
4. The system checks the information filled and acknowledges. [Alternate 1]

Exit Condition: The account will be added.

Alternate Flow 1 [if the administrator fills the existed user name/if the administrator fills wrong data]

1.1 The system displays the wrong message dialogue and resets the Create User Account form empty.

❑ **Quality requirements**

The system will create a new account with in reasonable amount of time depending on the system work load.

USE CASE SCENARIOS

- A use case is an abstraction that describes all possible scenarios involving the described functionality.
- A scenario is an instance of a use case describing a concrete set of actions.
- Scenarios are used as examples for illustrating common cases; their focus is on understandability.
- Use cases are used to describe all possible cases; their focus is on completeness.

We describe a scenario using a template with three fields:

- ❑ **Name:**
  - ✓ The name of the scenario enables us to refer to it unambiguously. The name of a scenario is <u>underlined</u> to indicate that it is an instance.
- ❑ **Participating actor instances:**
  - ✓ The participating actor instances field indicates which actor instances are involved in this scenario. Actor instances also have <u>underlined</u> names.
- ❑ **Flow of events :**
  - ✓ The flow of events of a scenario describes the sequence of events step by step.

Use Case Name: AddUserAccount

Participating Actor: Abebe

Description: Allow Abebe to add accounts.

Entry conditions: Abebe should log in.
Flow of Events:

1. Abebe initiates “Add User Accounts” use case.
2. The System displays the Create User form.
3. Abebe fills in the details of the new user like name, Id, role, user name, and password.
4. The system checks the information filled and acknowledges.

Exit Condition: The account will be added.

SAMPLE FORMAT FOR SRS

- Introduction
- Current System
- Proposed System
  - Overview
  - Functional Requirements
  - Non-Functional Requirements
- System models
  - Use case diagrams
  - Use case Description
  - Object model
  - Dynamic Model
  - User Interface components and navigation

VALIDATION

- The basic objective of the requirements validation activity is to ensure that the SRS reflects the actual requirements accurately and clearly.
- Helps to check that the SRS document is itself of "good quality"

❑ **Validation** is the process of confirming the completeness and correctness of requirements.

❑ Validation also ensures that the requirements:

- ✓ achieve stated business objectives,
- ✓ meet the needs of stakeholders, and
- ✓ are clear and understood by the developers.

❑ Validation is essential to identification of missing requirements and to ensure that the requirements meet certain quality characteristics.

- Validation should focus on uncovering errors of the following type:
  - Omission: *some user requirement is simply not included in the SRS*
  - Inconsistency: *contradictions within the requirements themselves*
  - Incorrect fact: *some fact recorded in the SRS is not correct.*
  - Ambiguity: *some requirements that have multiple meanings*

- Inspections of the SRS, frequently called requirements review, are the most common method of validation
- It is a review by a group of people to find errors and point out other matters of concern in the requirements specifications of a system.

- The group may include
  - author of the requirements document
  - someone who understands the needs of the client
  - a person of the design team, and the person(s) responsible for maintaining the requirements document.
  - some people not directly involved with product development, like a software quality engineer.

- Checklists are frequently used in reviews
- A checklist may include
  - Are all hardware resources defined?
  - Have the response times of functions been specified?
  - Have all the hardware, external software, and data interfaces been defined?

- Have all the functions required by the client been specified?
- Is each requirement testable?
- Are the responses to exceptional conditions specified?
- Does the requirement contain restrictions that can be controlled by the designer?
- Are possible future modifications specified?

REQUIREMENT VALIDATION

- Ad-hoc: The reviewers simply attempts to find as many defects as possible by examining the document using the skills and knowledge they have.
- Checklist-based: the reviewer is given a checklist with questions that are to be answered during the review.
  - The questions shall draw the attention of the reviewer to some aspects of the inspected document that are often found defective.
- Perspective-based reading: When using the perspective-based reading technique reviewers use different roles or points of view when reviewing.
  - For example, reviewing as a tester, reviewing as a designer, etc. Each role has scenarios that include questions and activities that tell the reader how to review.
- Scenario-based:
  - Defect-based reading: To use the defect-based reading technique we need to create a model of possible defects in requirements documents.
    - For each defect class from the model (e.g. the class of data type inconsistencies, incorrect functions, and/or missing or ambiguous functions), we develop a set of questions that would characterize the defect class.
    - While reading the document, the reviewer tries to answer the questions and find defects in the document.

DOCUMENTING REQUIREMENTS ELICITATION

- The results of the requirements elicitation and the analysis activities are documented in the SRS.
- This document completely describes the system in terms of functional and nonfunctional requirements.
- The audience for the SRS includes the client, the users, the project management, the system analysts (i.e., the developers who participate in the requirements), and the system designers (i.e., the developers who participate in the system design).