# Requirements Analysis

Lecture 2

Introduction to Requirements Analysis

UML Use Case Model or Use Case Diagram

## Introduction to Requirements

- Requirements are a description of how a system should behave or a description of system properties or attributes.

or

- It can alternatively be a statement of 'what' an application is expected to do.

or

- Requirements constitute system descriptions, or system capabilities or features that systems must have as well as constraints that systems must satisfy in order to be accepted by stakeholders.

## Why do we need requirements?

We use requirements for a variety of purposes, including:

- Project scoping
- Cost estimating
- Budgeting
- Project scheduling
- Software design
- Software testing
- Documentation and training manuals

## Functional Requirements

- Functional user requirements may be high-level statements of what the system should do or constitute the expected services of the system.
- Functional requirements refer to the functionality or services that the system is expected to provide, describing interactions between the system and its environment, while not focusing on implementation details.

## Example of Functional Requirements

- Requirements for an automated teller machine (ATM)
  - The ATM system shall check the validity of the inserted ATM card.
  - The ATM shall validate the PIN number enters by the customer
  - The ATM shall dispense no more than $600 against any ATM card in any 24hour period.

## Non-functional Requirements

- Non-functional requirements is a collective term adopted to describe aspects of the system (quality attributes and constraints) that are not directly related to the functional behaviour of the system.
- Generally non-functional requirements are captured into five categories:
  - Usability
  - Reliability
  - Performance
  - Supportability
  - Security

## Non-functional Requirements

- Usability describes the ease with which the system can be learned or used. A typical usability requirement might state:
  - The system should allow novice users to install and operate it with little or no training.
  - The end user shall be able to place an order within thirty seconds.
  - The end user shall be able to access any page within four seconds.
- Reliability describes the degree to which the system must work for users. Specifications for reliability typically refer to availability, mean time between failures, mean time to repair, accuracy, and maximum acceptable bugs. For example:
  - The system shall meet the terms of a Service Level Agreement.
  - The mean time to failure shall be at least four months.

## Non-functional Requirements

- Performance specifications typically refer to response time, transaction throughput, and capacity. For example:
  - All Web pages must download within three seconds during an average load, and five seconds during a peak load.
  - While executing a search, the system must be able to display 500 search results per page.
- Supportability refers to the software's ability to be easily modified or maintained to accommodate typical usage or change scenarios. Here are some examples of supportability requirements:
  - The system shall allow users to create new workflows without the need for additional programming.

## Non-functional Requirements

- Security refers to the ability to prevent and/or forbid access to the system by unauthorized parties. Some examples of security requirements are:
  - User authentication shall be via the corporate Single Sign on system.
  - Only authorized payroll administrators shall be permitted to access employee pay information.

## Requirements Analysis

- Requirements Analysis is the process of understanding the customer needs and expectations from a proposed system or application.
- The Requirements Analysis Process covers the complex task of eliciting and documenting the requirements of all the users, modeling and analyzing these requirements and documenting them as a basis for system design.

## Requirements Analysis

Conceptually, requirements analysis includes three types of activity:

- Eliciting requirements/Requirements gathering : the task of communicating with customers and users to determine what their requirements are.
- Analyzing requirements: determining whether the stated requirements are unclear, incomplete, ambiguous, or contradictory.
- Recording requirements: Requirements may be documented in various forms, such as natural-language documents, use cases, or process specifications.

## Techniques of Requirement Analysis

- Interviewing customers and domain experts
- Questionnaires
- Observation
- Study of documents
- Prototyping
- Brainstorming
- Use Cases

## Use Cases

- A use case is something an actor wants the system to do.
- A use case describes behavior that the system exhibits to benefit one or more actors.
- Use cases are always started by an actor or are always written from the point of view of actors.
- A use case consists of a series of actions that a user must initiate to carry out some useful work and to achieve his/her goal.

## Use Cases

- Use cases reflect all the possible events in the system in the process of achieving an actor's goal.
- A complete set of use cases specifies all the possible ways the system will behave, and defines all the requirements of the system.
- Use cases are used to capture functional requirements only.

## Use Cases

- Examining the goals the system supports makes the selection of use cases easier.

  *"Place an order."*

  *"Get money from bank account."*

  *"Get a quote."*

  *"Set up an advertising program."*

- Goals summarize system function in understandable, verifiable terms of use.

## Identifying Use Cases

Here is a helpful list of questions you can ask to identify use cases

- What functions will a specific actor want from the system?
- Does the system store or retrieve information?
- What happens when the system changes state ?
- Does the system interact with any external system?
- Does the system generate any reports ?

## Use Case Modeling

- A use case describes what a system does rather than how it does it.
- Use case analysis focus is not the internal view of the system or implementation details.
- Use case modeling is the process of describing the behavior of the target system from external point of view.

## Use Case Model

- Actors
- Use cases
- Four kinds of relationships
  - Association is between actor and use case
  - Include :Included use case is always necessary for the completion of the activating use case.
  - Extend : Another use case is activated occasionally at specific extension point.
  - Generalization: can generalize use cases when they achieve same goal by different means.

## Steps in Use Case Model Development

- Choose the system boundary.

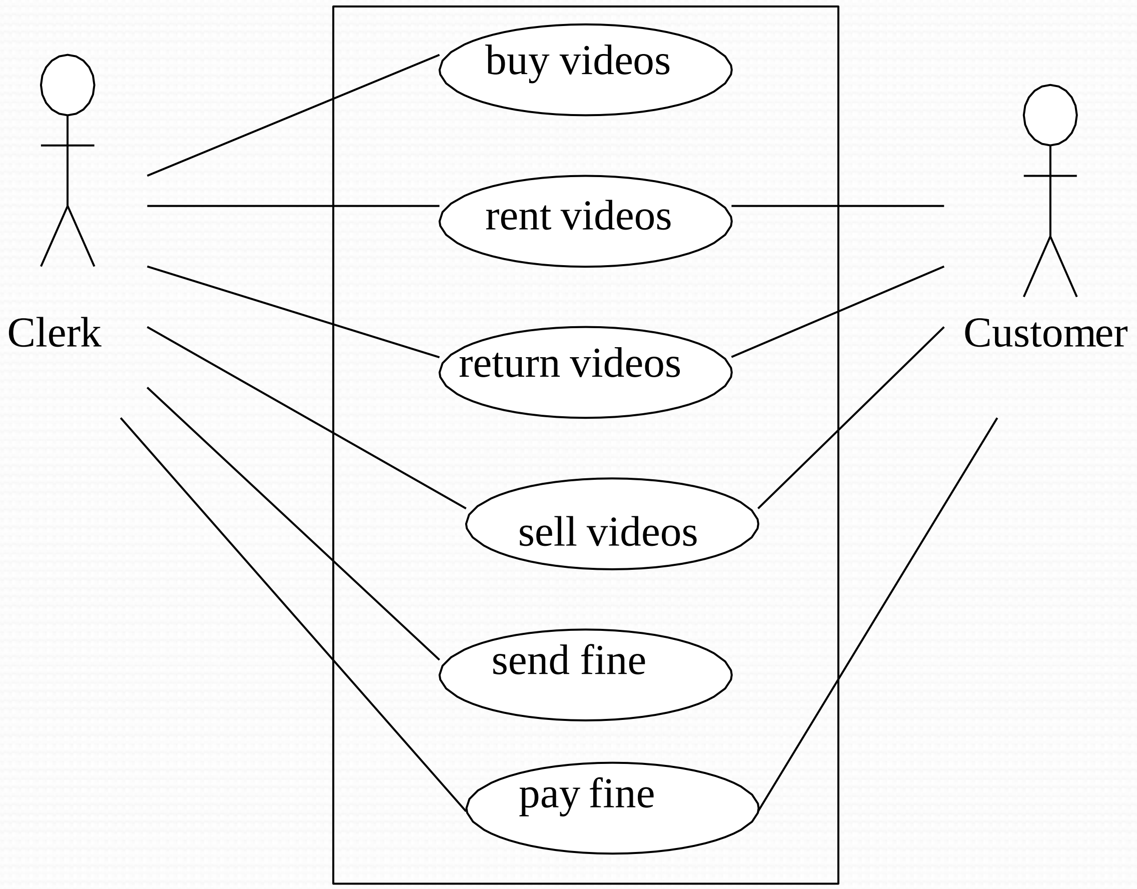
  You decide what is inside and what is outside of the system by drawing the system boundary.The system boundary is usually drawn as box.

- Identify actors. An actor is something with behaviour, such as a person, computer system, or organization, e.g. a cashier. An actor is a labelled figure (a person) or box (a system) outside the box or system boundary.
  - Primary actor: is the actor whose goal identifies and drives the use case.
  - Secondary actor: that the system needs assistance from to achieve the primary actors goal.

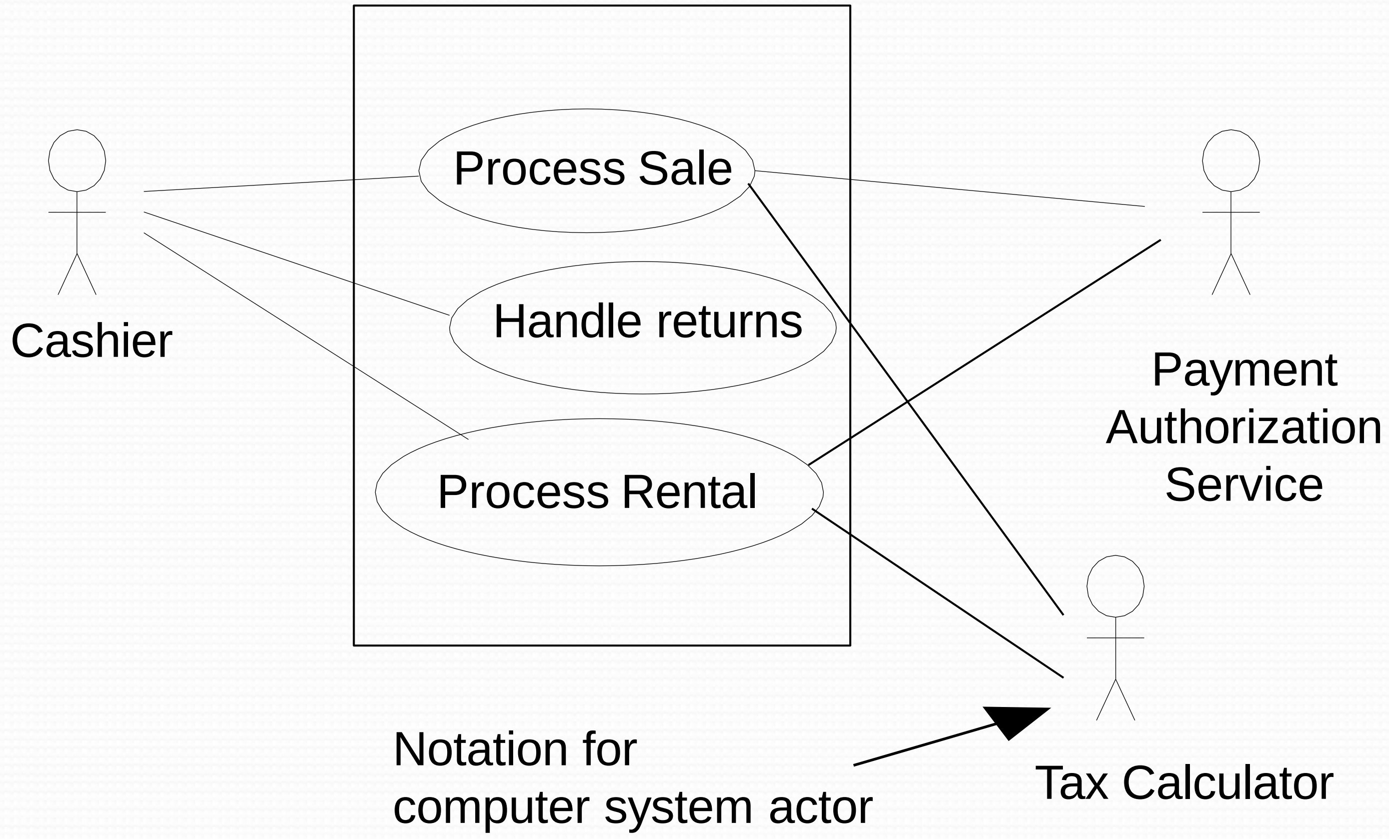
## Steps in Use Case Model Development

- For each actor identify their user goals. Define use cases that satisfy user goals; name them according to their goal.
    - The name of a use case starts with a verb, because a goal is something to do. A name has the form <verb object>, such as <rent videos>. Use specific verbs such as "rent"; do not use generic verbs such as "do"
- A use case is a labelled ellipse inside the box.
- A link or line between an actor and a use case means the actor participates in the use case.

## Use Case Model (Video System)



## Use Case Model (Process Sale)


Process Sale
Handle returns
Process Rental
Cashier
Payment Authorization Service
Tax Calculator
Notation for computer system actor

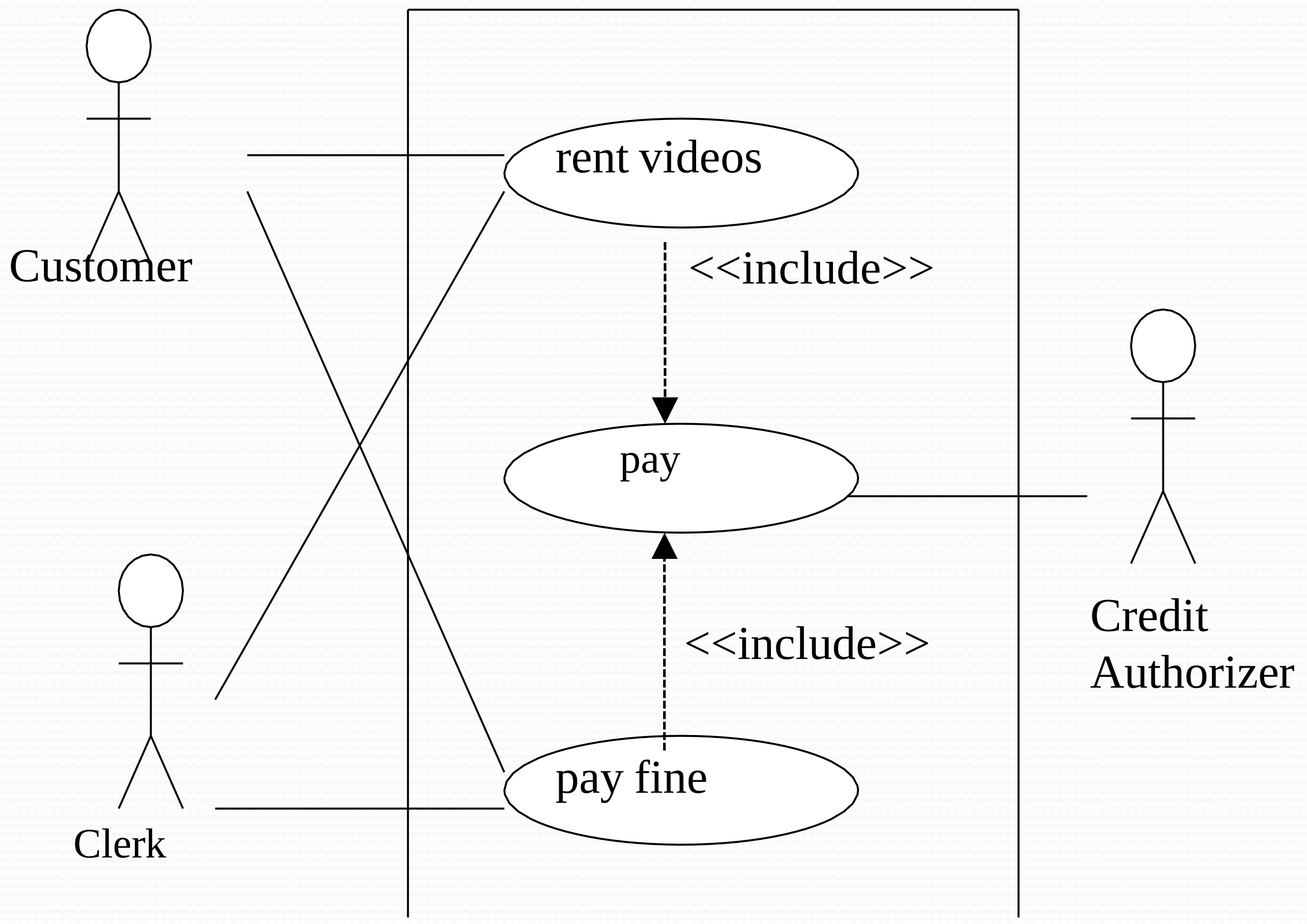
## Incremental Development of Use Case Model

- Not all requirements in the Process Sale are shown in this iteration or model.
- It is common to work on varying scenarios or features of the same use case over several iterations and gradually extend the system to ultimately handle all the functionality required.
- On the other hand, short, simple use cases on the core functionality may be completed within one iteration.

## Include Relation

- The include relation is used to show that the same set of actions are included in several use cases.
- The included actions are placed in a separate oval and linked to the essential use cases with an <<include>> link.
- The included actions are shown as a use case, and linked to any actors unique to those actions; the other actors are linked to the primary use case.
- An arrow labelled <<include>> is drawn from the primary cases to the included case.

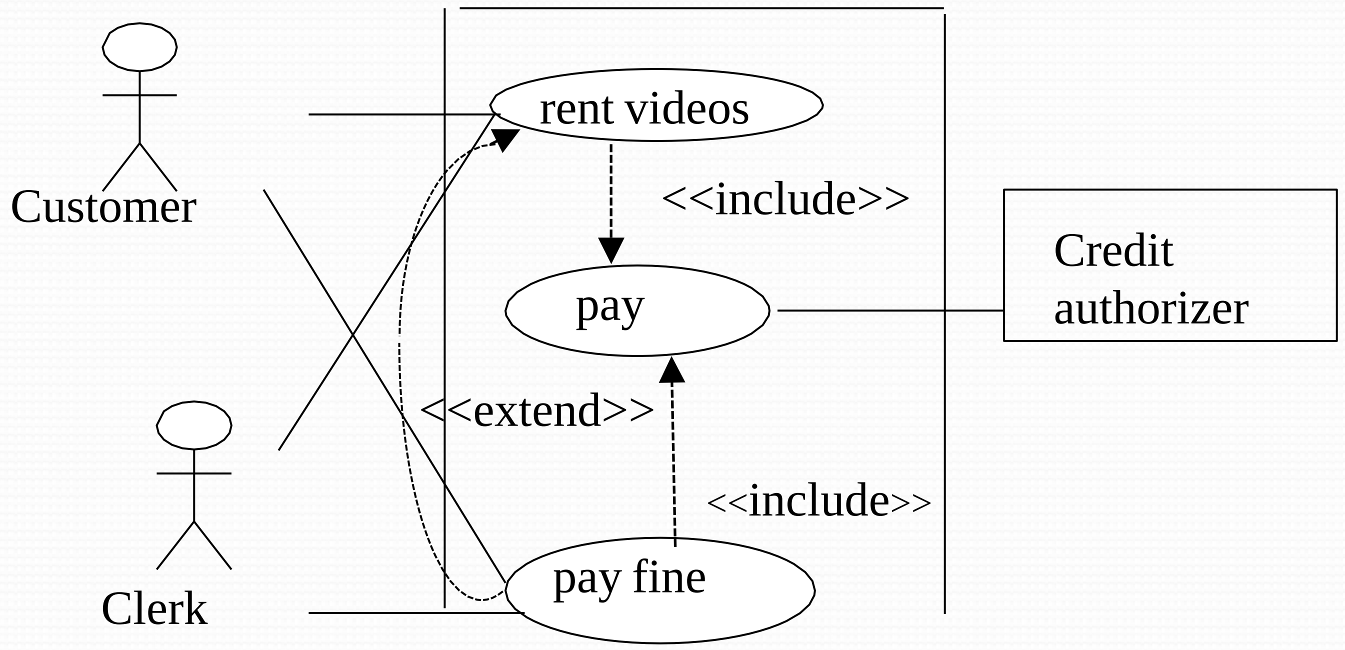
## Include Relation



## Extend Relation

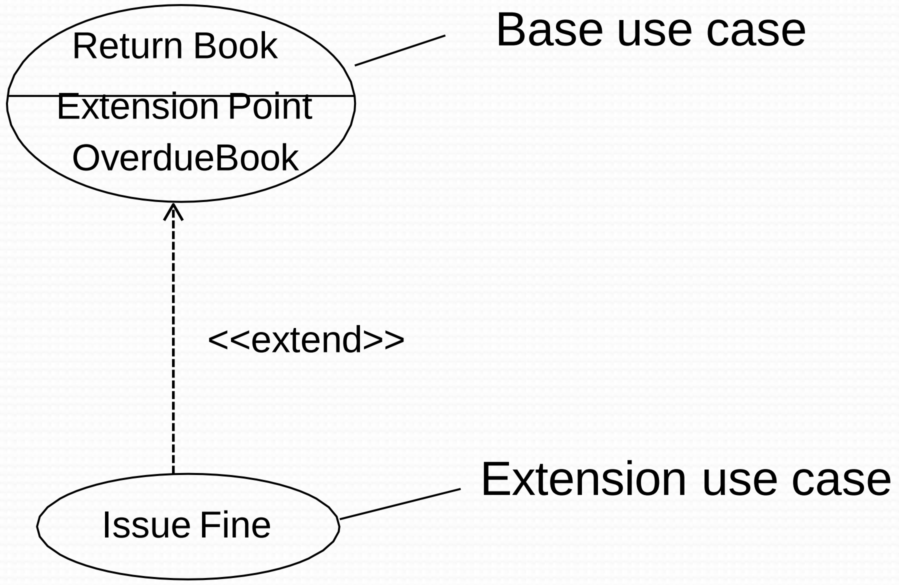
- The extend relation is used to show that a set of actions sometimes occur in a use case; the actions do not always have to be executed as part of the use case
- The actions are shown as a separate use case, with an arrow pointing towards the primary use case labelled <<extend>>.
  - A video store customer may simply come in to pay a fine, so that is a primary use case. The store does not rent videos to a customer with an unpaid fine, so a customer may be asked to pay the fine when they try to rent. This happens only when there are unpaid fines so it is a "sometimes" use case; "pay fine" extends "rent video".

## Extend Relation



# Extend Relation

- Base use case does not do anything about extension use cases –it just provides hooks for them. In fact,the base use case is complete without extensions.

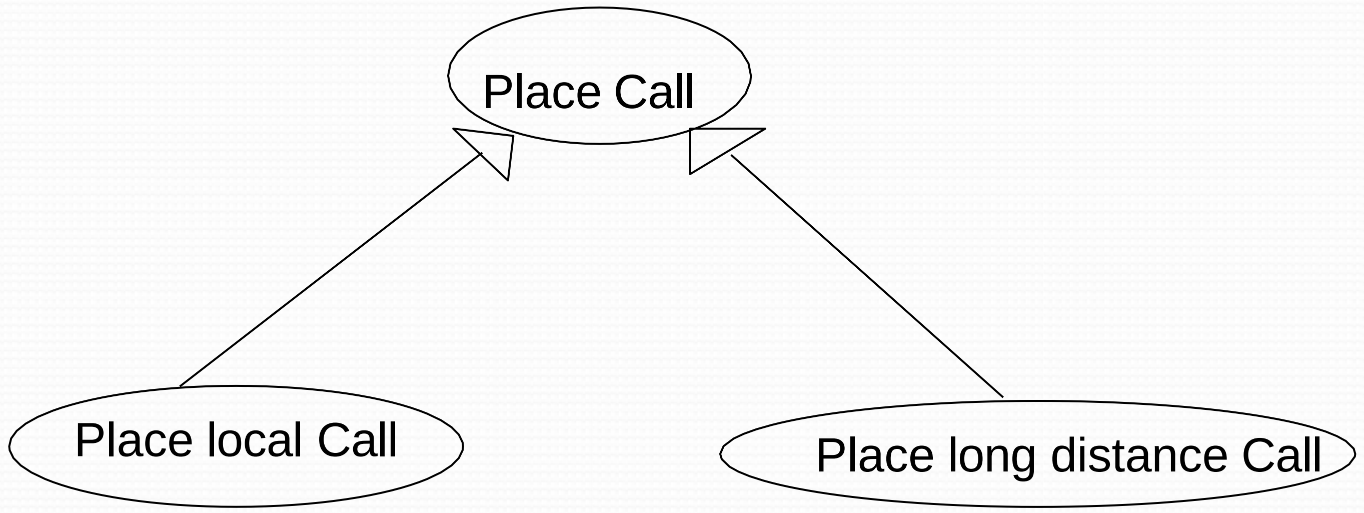


## Use Case Generalization

- Generalization is used when you find two or more use cases that have commonalities in behavior, structure, and purpose.
- Describe the shared parts in a parent use case, that is then specialized by children use cases.
- The children use case may
  - Inherit feature from their parent use case
  - Add new features
  - Change inherited features

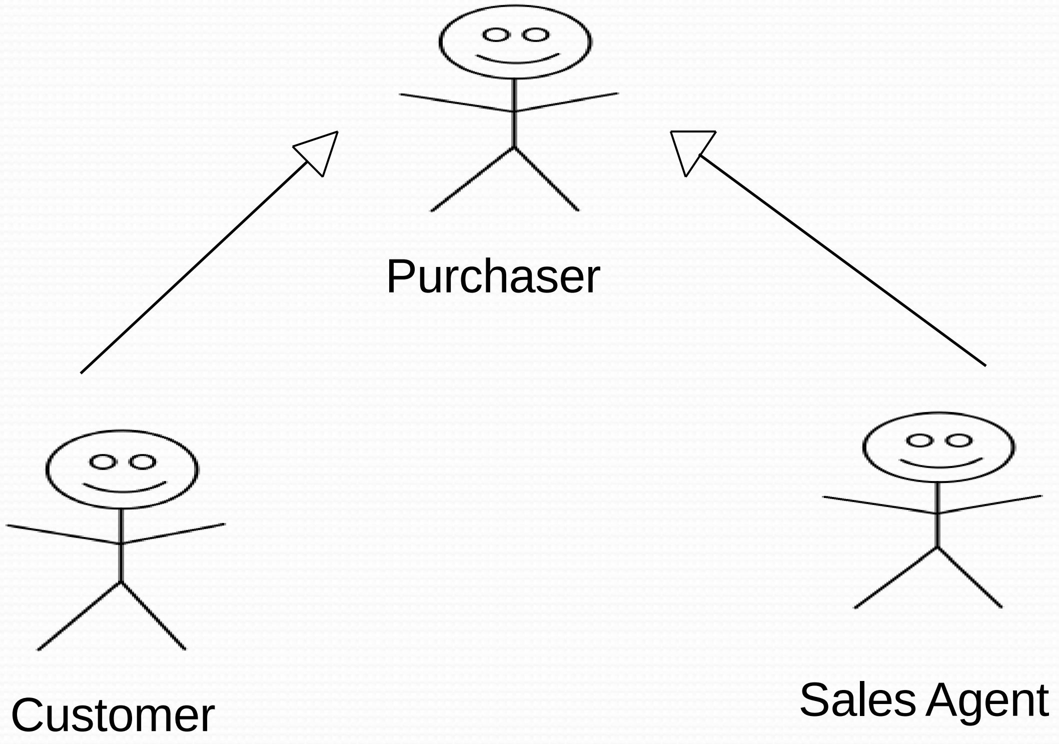
## Use Case Generalization

Use case generalization factors out behavior common to one or more use cases into a parent use case.



## Actor Generalization

- Actor generalization factors out behavior common to two or more actors into a parent actor.



## CRUD Operations

- A business system is built on four basic operations that are performed by a database:
  - **C**reate a record, such as a customer record
  - **R**etrieve the record given a (usually unique) key
  - **U**pdate the record with new data and store it
  - **D**elete a record

These are known as the CRUD operations.

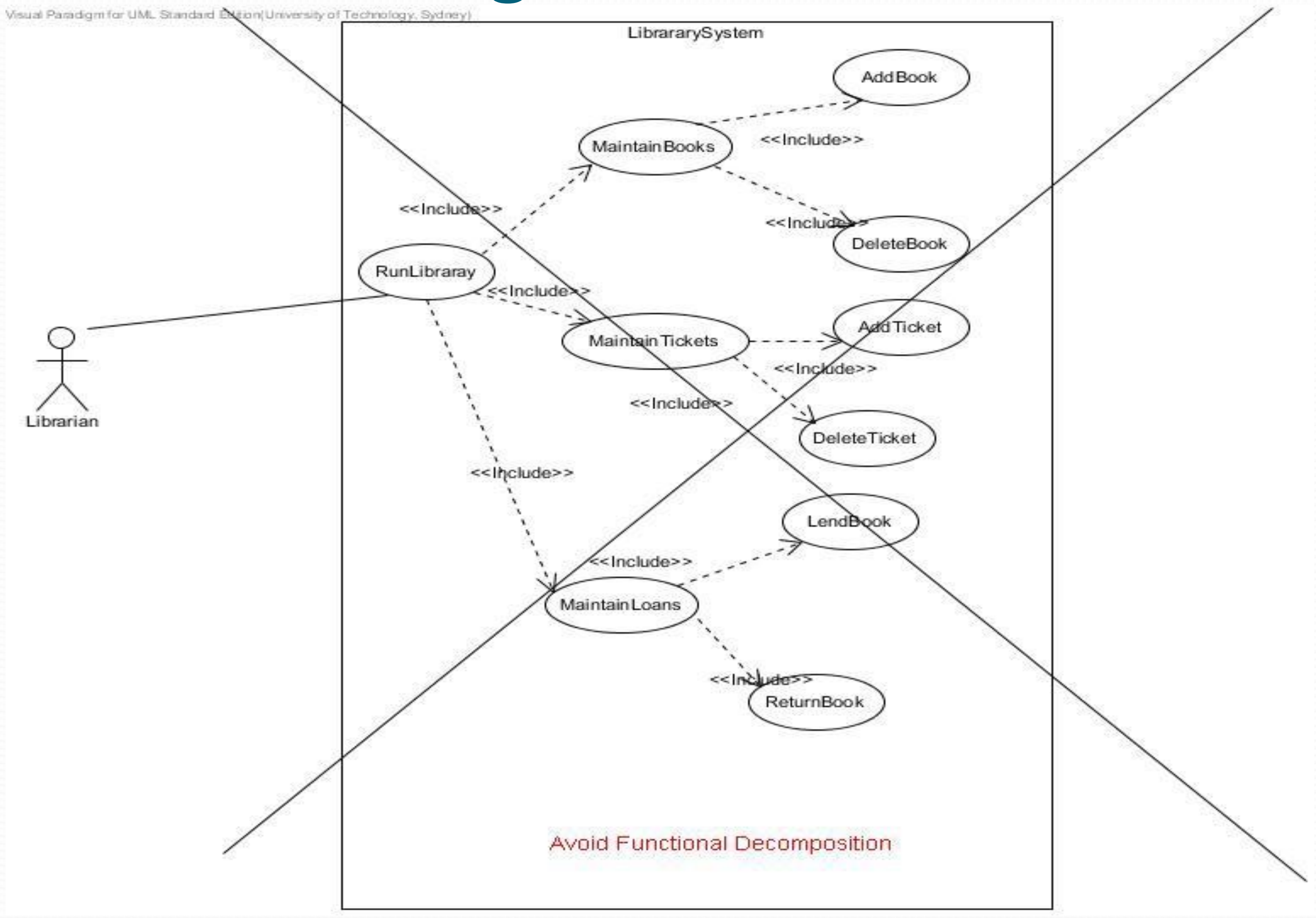
## CRUD Use Cases

- Uses Cases of the sort *Create an Item*, *Retrieve an Item*, *Update an Item*, *Delete an Item*.
- In principle, they are separate because each is separate goal, possibly carried by a different person with a different security level.
- However, they clutter up the use case set and can triple the number of items to track.
- Trend is to just start with one use case, *Manage an Item*. If description becomes more complex break out the use case.

## Hints for Modeling Use cases

- Keep use cases short and simple
  - A good rule of thumb is to ensure that the main flow of a use case fits on a single side of paper
- Focus on what, not the how
  - Remember that you are writing use cases to work out what the actors need the system to do, not how the system should do it.
- Avoid functional decomposition
  - One common error in use case analysis is to create a set of “high level” use cases and then break these into lower level use cases.

## Hints for Writing Use cases



## Further Reading

- Chapter 4, 6 &7, Object-Oriented Systems Analysis and Design. Noushin Ashrafi and Hessam Ashrafi
- Geri Schneider and Jason P. Winters. Applying Use Cases, Addison-Wesley, 2001
- ISO/IEC 9126 or ISO/IEC 25010 : International standard for the evaluation of software quality