Object-Oriented Design

Lecturer: Raman Ramsin

Lecture 5: Use Case Modeling
Part 2

Activities of requirements workflow

Capture Functional Requirements

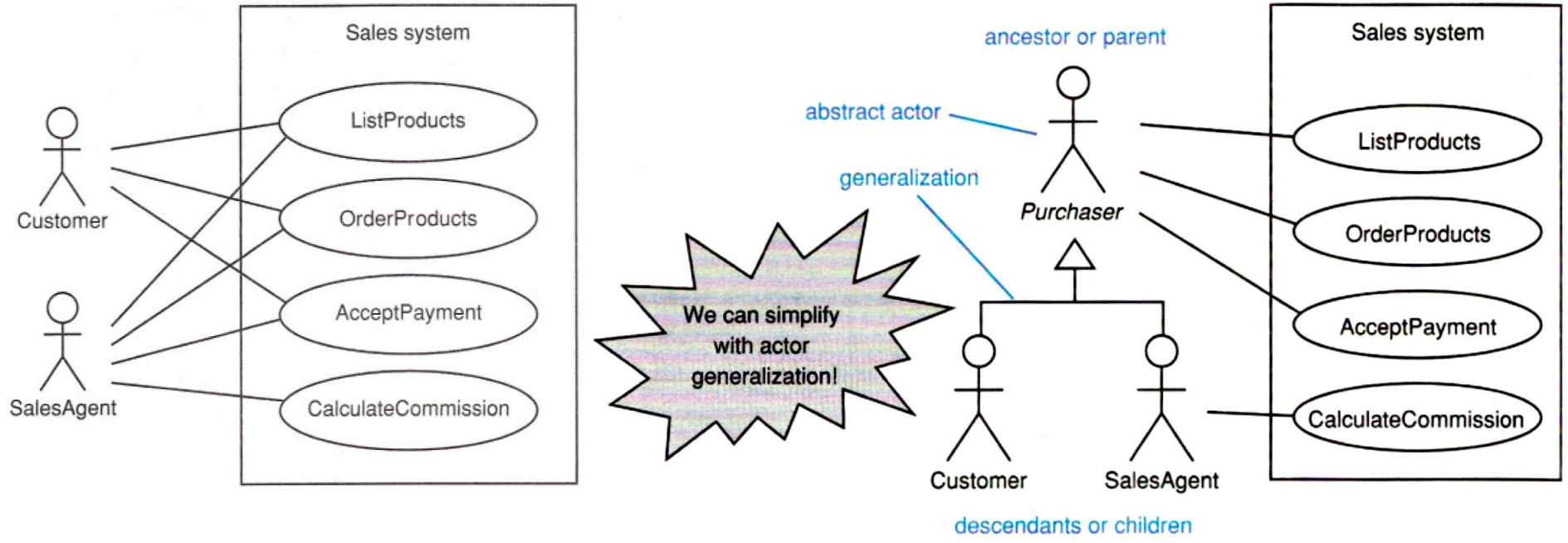
1. **Find actors and use cases**
2. Prioritize use cases
3. **Detail use cases**
4. Prototype user interface
5. ***Structure the use-case model***

Relationships

- *actor generalization* - a generalization relationship between a more general actor and a more specific actor
- *use case generalization* - a generalization relationship between a more general use case and a more specific use case
- *«include»* - a relationship between use cases that lets one use case include behavior from another
- *«extend»* - a relationship between use cases that lets one use case extend its behavior with one or more behavior fragments from another

Actor Generalization

- If two actors communicate with the same set of use cases in the same way, we can express this as a generalization to another (possibly abstract) actor.



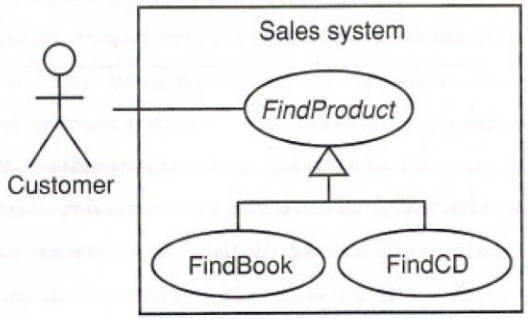
Use Case Generalization

- Used when one or more use cases are really specializations of a more general case.
- Child use cases may:
  - *inherit* features from their parent use case
  - *add* new features
  - *override* (change) inherited features (except for inherited relationships and extension points)
- Use case generalization is documented in use case specifications using a simple tag language on flow steps. There are two rules:
  - Each step number in the child is postfixed by the equivalent step number in the parent if there is one.
  - If the step in the child overrides a parent step, it is postfixed by "o" (for overridden) and then the step number in the parent.

Use Case Generalization: Features

Use case feature	Inherit	Add	Override
Relationship	Y	Y	N
Extension point	Y	Y	N
Precondition	Y	Y	Y
Postcondition	Y	Y	Y
Step in main flow	Y	Y	Y
Alternative flow	Y	Y	Y

Use Case Generalization: Example

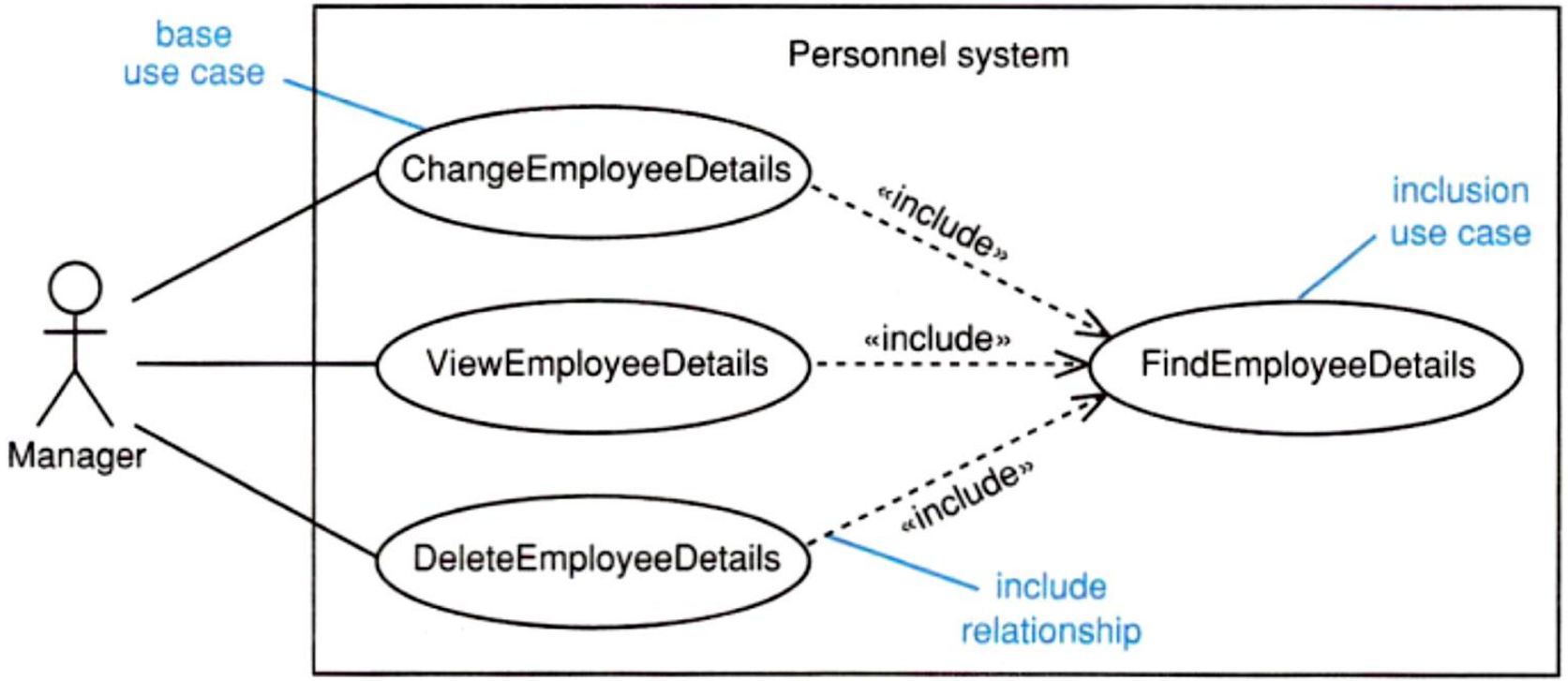


Use case: *FindProduct*
ID: 6
Brief description: The Customer searches for a product.
Primary actors: Customer
Secondary actors: None.
Preconditions: None.
Main flow: 1. The Customer selects "find product". 2. The system asks the Customer for search criteria. 3. The Customer enters the requested criteria. 4. The system searches for products that match the Customer's criteria. 5. If the system finds some matching products 5.1 The system displays a list of the matching products. 6. Else 6.1 The system tells the Customer that no matching products could be found.
Postconditions: None.
Alternative flows: None.

	Use case: FindBook
	ID: 7
	Parent ID: 6
	Brief description: The Customer searches for a book.
	Primary actors: Customer
	Secondary actors: None.
	Preconditions: None.
	Main flow:
overridden	1. (o1.) The Customer selects "find book".
overridden	2. (o2.) The system asks the Customer for book search criteria comprising author, title, ISBN, or topic.
inherited without change	3. (3.) The Customer enters the requested criteria.
overridden	4. (o4.) The system searches for books that match the Customer's criteria.
overridden	5. (o5.) If the system finds some matching books
added	5.1 The system displays the current best seller.
overridden and renumbered	5.2 (o5.1) The system displays details of a maximum of five books.
added	5.3 For each book on the page the system displays the title, author, price, and ISBN.
added	5.4 While there are more books, the system gives the Customer the option to display the next page of books.
inherited without change	6. (6.) Else
added	6.1 The system displays the current best seller.
renumbered	6.2 (6.1) The system tells the Customer that no matching products could be found.
	Postconditions: None.
	Alternative flows: None.

«include» Relationship

- The «include» relationship between use cases allows you to include the behavior of one use case into the flow of another use case.



«include» Relationship: Specification

- The base use case is not complete without all of its inclusion use cases.
- Inclusion use cases may be complete (*instantiable*) or incomplete (*behavior fragments*)

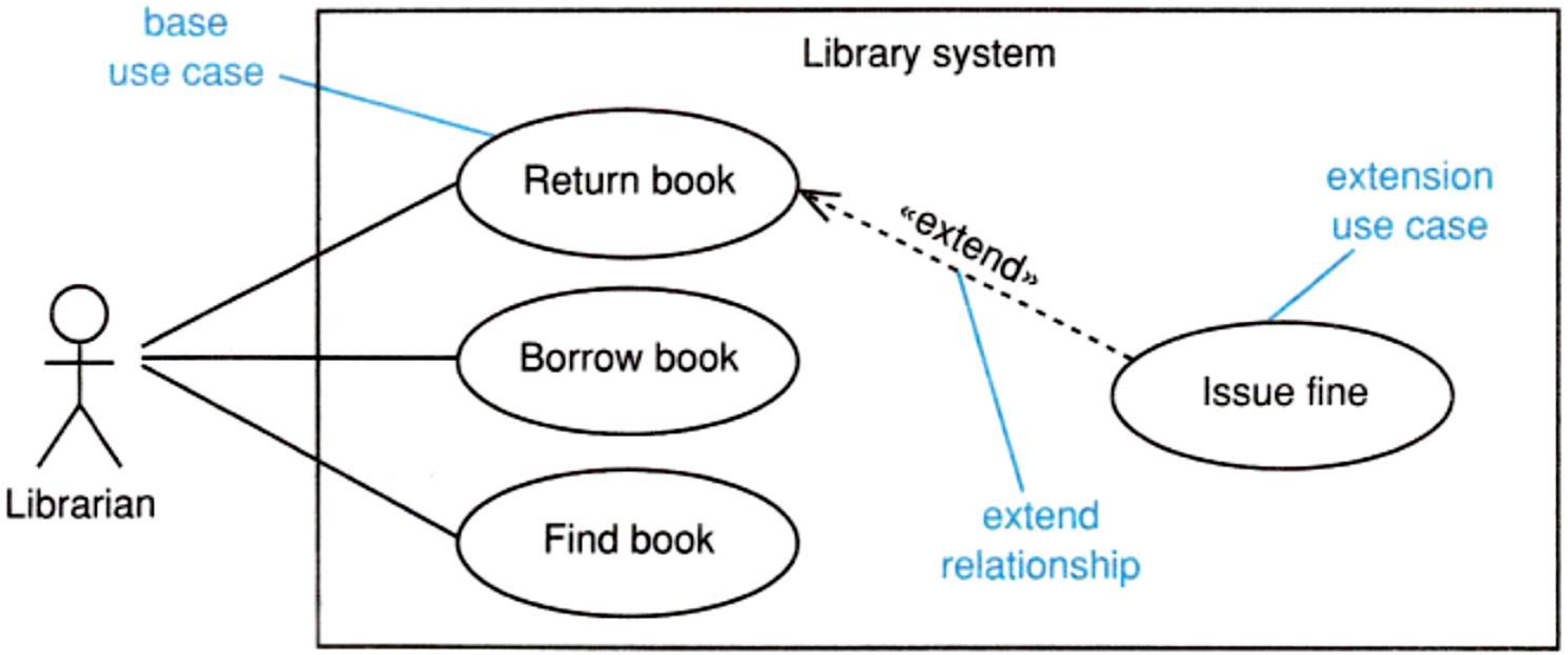
Use case: ChangeEmployeeDetails
ID: 1
Brief description: The Manager changes the employee details.
Primary actors: Manager
Secondary actors: None.
Preconditions: 1. The Manager is logged on to the system.
Main flow: 1. include(FindEmployeeDetails). 2. The system displays the employee details. 3. The Manager changes the employee details. ...
Postconditions: 1. The employee details have been changed.
Alternative flows: None.

Use case: FindEmployeeDetails
ID: 4
Brief description: The Manager finds the employee details.
Primary actors: Manager
Secondary actors: None.
Preconditions: 1. The Manager is logged on to the system.
Main flow: 1. The Manager enters the employee's ID. 2. The system finds the employee details.
Postconditions: 1. The system has found the employee details.
Alternative flows: None.

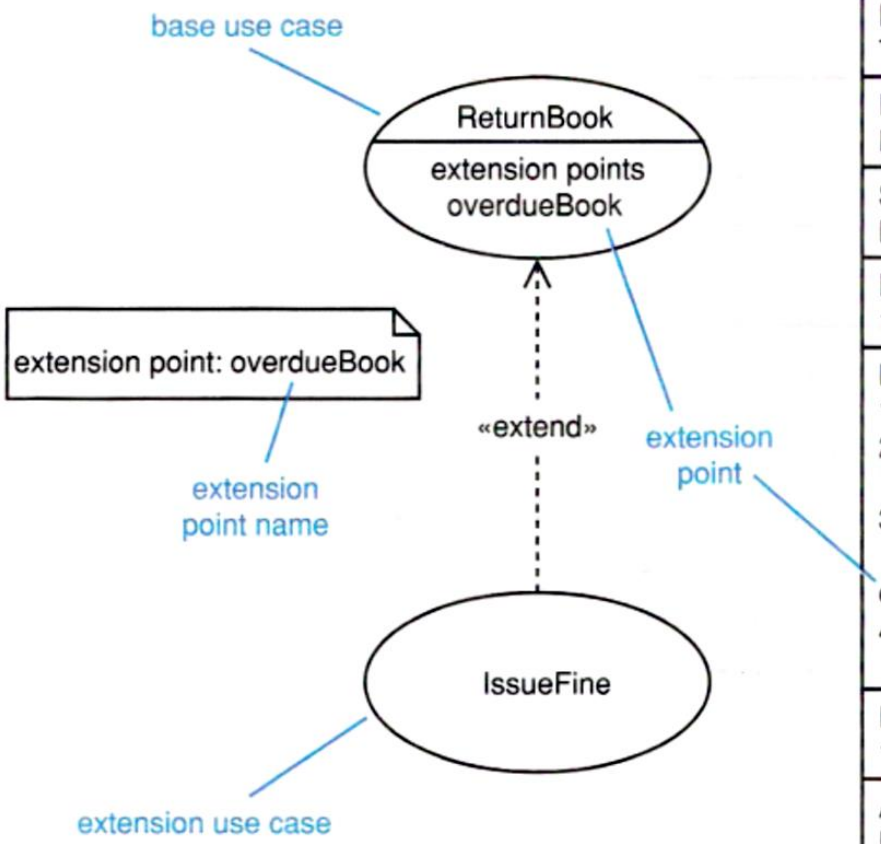
«extend» Relationship

- provides a way to insert new behavior into an existing use case.
- The *base use case* is a complete use case and provides a set of *extension points* that are hooks where new behavior may be added.
- The *extension use case* provides a set of *insertion segments* that can be inserted into the base use case at these hooks.
- The relationship can specify *exactly* which extension points in the base use case are being extended.

«extend» Relationship: Example



«extend» Relationship: Extension Points

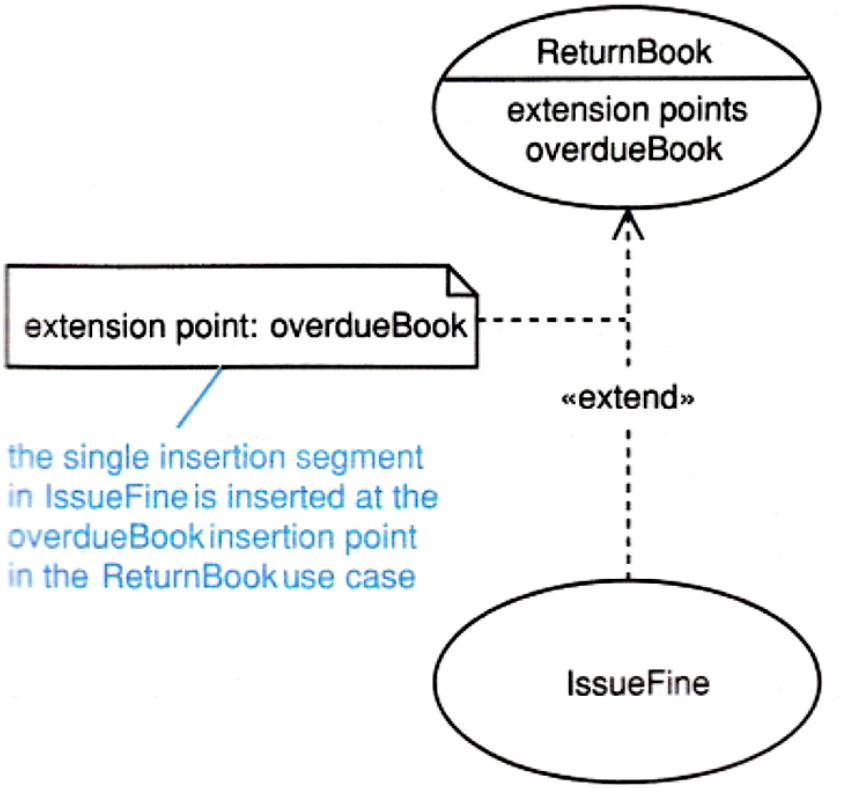


Use case: ReturnBook
ID: 9
Brief description: The Librarian returns a borrowed book.
Primary actors: Librarian
Secondary actors: None.
Preconditions: 1. The Librarian is logged on to the system.
Main flow: 1. The Librarian enters the borrower's ID number. 2. The system displays the borrower's details including the list of borrowed books. 3. The Librarian finds the book to be returned in the list of books. extension point: overdueBook 4. The Librarian returns the book. ...
Postconditions: 1. The book has been returned.
Alternative flows: None.

«extend» Relationship: Extension Use Cases

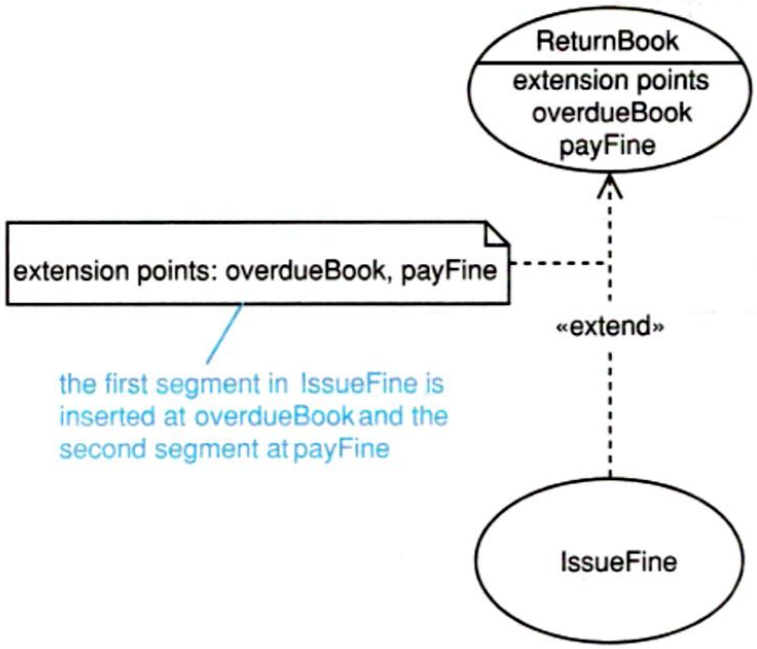
- Are not complete use cases and therefore can't be instantiated.
- Normally consist of one or more behavior fragments known as *insertion segments*.
- The «extend» relationship specifies the extension point in the base use case where the insertion segment will be inserted.

Extension Use Case: Single-Segment Example



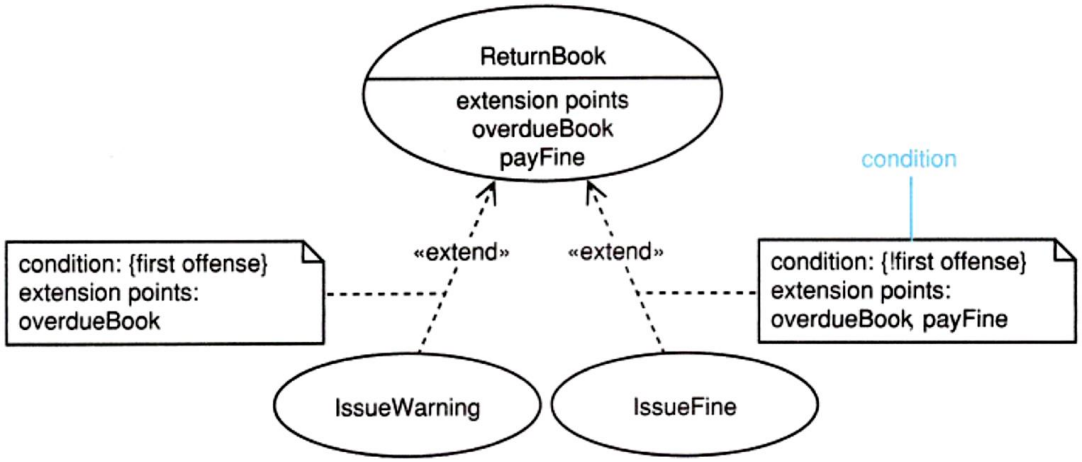
Extension Use case: IssueFine
ID: 10
Brief description: Segment 1: The Librarian records and prints out a fine.
Primary actors: Librarian
Secondary actors: None.
Segment 1 preconditions: 1. The returned book is overdue.
Segment 1 flow: 1. The Librarian enters details of the fine into the system. 2. The system prints out the fine.
Segment 1 postconditions: 1. The fine has been recorded in the system. 2. The system has printed out the fine.

Extension Use Case: Multiple-Segment Example



Extension Use case: IssueFine
ID: 10
Brief description: Segment 1: The Librarian records and prints out a fine. Segment 2: The Librarian accepts payment for a fine.
Primary actors: Librarian
Secondary actors: None.
Segment 1 preconditions: 1. The returned book is overdue.
Segment 1 flow: 1. The Librarian enters details of the fine into the system. 2. The system prints out the fine.
Segment 1 postconditions: 1. The fine has been recorded in the system. 2. The system has printed out the fine.
Segment 2 preconditions: 1. A fine is due from the borrower.
Segment 2 flow: 1. The Librarian accepts payment for the fine from the borrower. 2. The Librarian enters the paid fine in the system. 3. The system prints out a receipt for the paid fine.
Segment 2 postconditions: 1. The fine is recorded as paid. 2. The system has printed a receipt for the fine.

«extend» Relationship: Conditional Extensions



Extension Use case: IssueWarning
ID: 11
Brief description: Segment 1: The Librarian issues a warning.
Primary actors: Librarian
Secondary actors: None.
Segment 1 preconditions: 1. The returned book is overdue.
Segment 1 flow: 1. The Librarian enters details of the warning into the system.
Segment 1 postconditions: 1. The warning has been recorded in the system.

«extend» Relationship: Rules

- The «extend» relationship must specify one or more of the extension points in the base use case or it is assumed that the «extend» relationship refers to *all* extension points.
- The extension use case must have the same number of insertion segments as there are extension points specified in the «extend» relationship.
- It is legal for two extension use cases to «extend» the same base use case at the same extension point. But if this happens, the order in which the extensions execute is indeterminate.

Use Case Modeling: Hints and Tips - 1

1. **Keep use cases short and simple**
   - Include only enough detail to capture the requirements.
   - A good rule of thumb is to ensure that the main flow of a use case fits on a single side of paper.
     - Start by simplifying the text
     - Remove any design details
     - Reanalyze the problem: Are there more than one use case? Can alternative flows be factored out?

Use Case Modeling: Hints and Tips - 2

2. Focus on the *what*, not the *how*

- Use cases should show *what* the actors need the system to do, not *how* the system should do it. The how comes later in the design workflow.
  - Example:

    .....
    4. The system asks the Customer to confirm the order.
    5. The Customer presses the OK button.
    .....

  - Some sort of user interface has been imagined: a form with an OK button.
  - Better be written as:

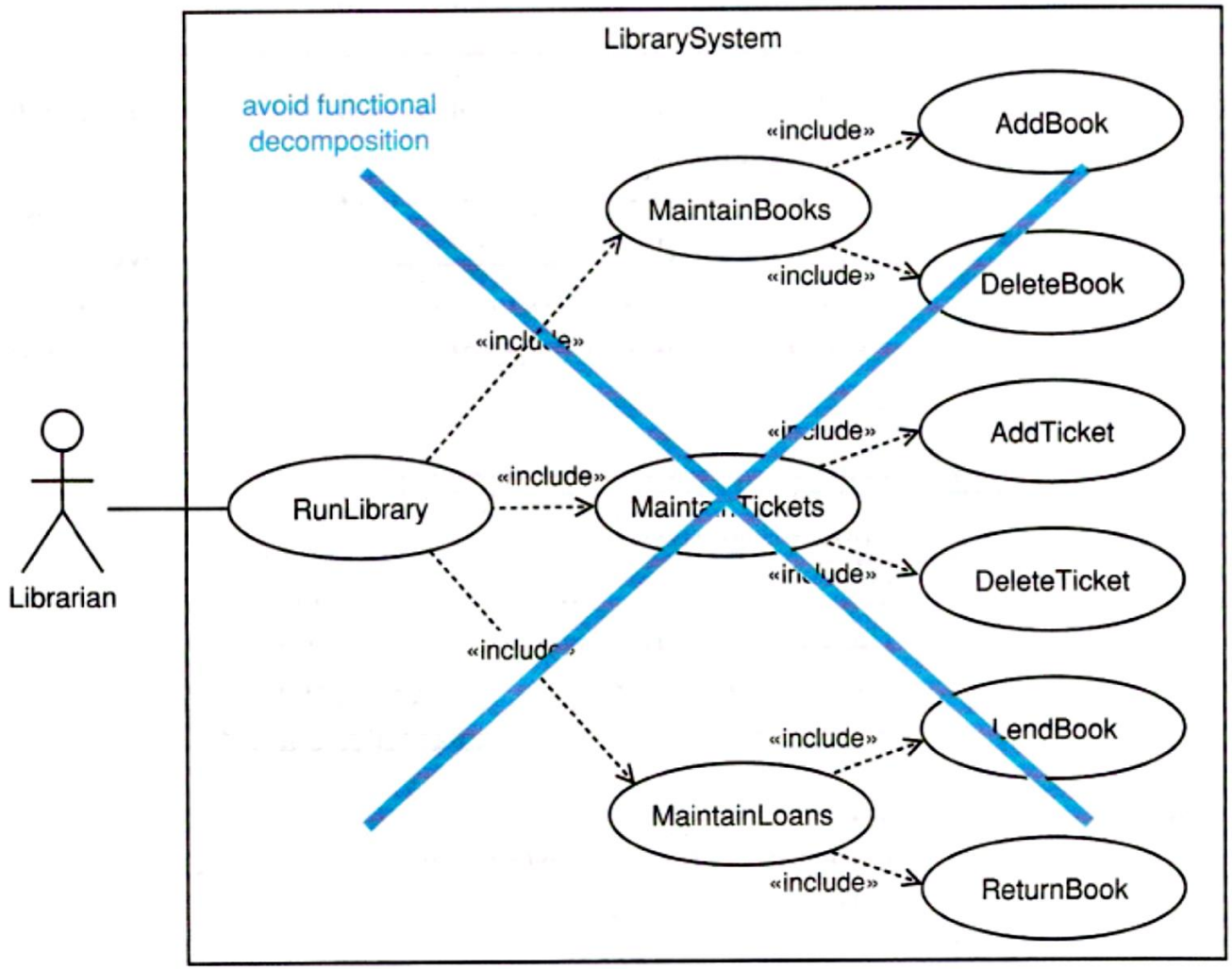
    5. The Customer accepts the order.

Use Case Modeling: Hints and Tips - 3

3. **Avoid functional decomposition**

- *Functional Decomposition* in Use Case Modeling: creating a set of "high level" use cases, and then breaking these down into a set of lower-level use cases and so on, until "primitive" use cases are reached that are detailed enough to be implemented.

Functional Decomposition in Use Case Modeling: Example



Reference

- Arlow, J., Neustadt, I., *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*, 2nd Ed. Addison-Wesley, 2005.