# MechEng SE3 Lectures 5 and 6 Use Cases in Detail

Simon Gay (slides by Ray Welland)

5 / 10 February 2010

## Project Description

- We are developing software to manage the employment, training and payment of part time teaching staff at a university or other such educational or training establishment.
- Before the start of each term or semester, the course directors produce a list of teaching requirements which we must try and fill. Our administrator will then attempt to find suitable staff, organise training for them and ensure that they are paid for the teaching they do.

# Use Cases (first attempt)

- Request teachers
- Recruit teachers
- Organise training
- Arrange payment

- *Assign Teacher to Duty*
- *Handling absences*

## Project Description (2)

- All teaching requests must be approved by the PTT Director, who will also have the facilities to check the budget. Our system will allow Course Directors to enter requests, and will store them for approval by the PTT director. It is expected that informal discussions will take place outside this system before formal requests are made via the system. [ Added after considering the role of the PTT director and budgetary control].
- After teaching starts, we will receive claim forms, which we pass to the finance department for payment; these claims must match the budget.

## Project Description (3)

- Sometimes teachers will not turn up, as detected by the course director. We will make sure that they are not paid and that they are informed of this. In other cases, teachers will request a replacement by a named individual or inform someone of a pending absence. The recruiter may well find a replacement. This process will take place outside our system, but the recruiter will enter any replacement or absence into the system. [ Added after considering the absence use cases. ]
- The claims forms must conform with the current finance department procedures (signed paper copies). Our system will record the approval of any payments and provide the facilities for checking if a payment is due. Any irregularities in payment will be handled outside the system. [Added after considering the payment use cases].

## Project Description (4)

- All teachers will be given an automatic warning that it is time to submit their next claim. [Added after considering any Timer based use cases.]
- The recruitment process takes place outside our system, we just provide facilities for teachers and teaching duties to be entered. [Added after considering the "Assign teacher to duty" use case].
- The system will maintain a record of teacher skills and training. The actual training will not be part of the system. [Added after the training use cases].

## Project Description (5)

- The system should store two types of data, current data referring to courses that are about to run or running, and historical data of past courses and teachers. Historical data will be purged after 5 years. [ Added after considering the remove use cases. ]
- The system will maintain create, update and read permissions for all information stored. [Added after considering use case details and associated actors.]

## Actors

- Course Director: a person who identifies a part time teaching need
- Recruiter: finds people to do the teaching
- (part-time) Teacher: does the teaching
- Payer: organises the payment
- Trainer:  trains the teachers
- PTT Director: co-ordinates PTT activities

## Use Cases

- Request Teachers
- OK Teaching Request
- Enter New Teacher
- Assign Teacher to Duty
- Cancel Assignment
- Check Teaching Assignments, Vacancies, Payments and Budget
  [ This is a consolidation of a number of different check use cases that appeared early in the design process. ]
- Inform of No Show
- Inform of Replacement
- Inform of Absence

## Use Cases (2)

- Check Teacher Skills
- Update Teacher Training Record
- Approve Payment
- Check Budget
- Enter Payment Rates
- Maintain Staff List
- Maintain History Information
- Add Deadline (for claim)
- Deadline Approaching

## Non Functional Requirements

- The user interface must be a GUI
- The system must run on a number of different types of machine, inc. Windows and Linux
- Different  levels of security must be implemented for different users:
  – Administrator, Course Director, Teacher
- Must be available during normal working hours

## Initial Plan

- The aim here is to prioritise activities.
- Risks
  - HIGH: administrative staff may reject the interface
  - LOW: other staff may reject the interface
  - MEDIUM: database may not always be available
  - MEDIUM: we may have missed undocumented procedures

# Use Cases

- ***Must Have***
  - Request Teachers
  - OK Teaching Request
  - Enter New Teacher
  - Assign Teacher to Duty
  - Cancel Assignment
  - Check Teaching Assignments and Vacancies

## Use Cases (2)

- ***Should Have***
  - Inform of No Show
  - Inform of Replacement
  - Inform of Absence
  - Approve Payment
  - Enter Payment Rates
  - Check Payments and Budget
  - Maintain Staff List

## Use Cases (3)

- ***Could Have***
  - Check Teacher Skills
  - Update Teacher Training Record

- ***Would Like to Have***
  - Maintain History Information
  - Add Deadline
  - Deadline Approaching

## Plan of Action

- We need a prototype to help us design the administrative interface. The key activity is entering and updating the assignment of teachers to duties and so the prototype should implement the Assign Teacher to Duty use case.
- A suitable prototyping tool would be Visual Basic or Visual C++ on a PC.
- We will implement all of the Must Have use cases in version 1 and Should Have cases in version 2.

## Detailing Use Cases

- The use case diagram provides a useful overview of the system functionality, and has helped us to refine the project description and understand the system boundary.
- The next step is to go in to more detail for each use case.
- We will learn more about the system functionality and begin to be in a position where we can start to design the system.
- The most important details that we want to discover are the flow of events.

## Scenarios

- A scenario is a single example of a use case being used.
- The scenario will use actual sample data, rather than general terms.
  - Do not say assign a teacher to a duty
  - Do say “Assign Ron to the Tuesday 3-5 level 1 lab.”
- There will be many different samples, describing different ways that the use case can go.
- The different scenarios can have different outcomes.
- One aim of the requirements gathering phase is to try to list all possible scenarios for each use case.

## Alternatives

- A scenario cannot have any if or loop statements, but the use case, which is the combination of all the scenarios, probably will.
- The “Request Teachers” scenario can lead to a successful request being made.
- Alternatively, the course director may make mistakes in the details, which can be verified by our system, leading to alternate scenarios.

## The Primary Scenario

- There will be one main scenario where everything goes right. This is the reason why the use case was generated.
- This is often called the *happy day* scenario.
- Secondary scenarios often involve exceptional circumstances, such as requesting teaching for a Sunday Lab.
- It is not always necessary to write up every secondary scenario in detail.
  - Complicated secondary scenarios do need to be detailed.
  - But we do not want to write the whole of the system in English first, before programming it later.
- It is often sufficient to list the names of all the secondary scenarios.

## Flow of Events

- There are several different styles for writing use case details. The most common are
  - Informal text.
  - Numbered steps.
  - Activity diagrams.
- Here are some of the use cases for the PTT example, written in the numbered steps style.

## Request Teachers Primary Scenario

- Rob accesses the system and brings up course 1.
- He enters the lab for Tuesday 10-12 in term 1.
- He enters the need for a tutor and an undergraduate demonstrator.
- He enters the lab for Wednesday 3-5 in term 1.
- He enters the need for a tutor and an undergraduate demonstrator.
- He suggests Jeremy as the tutor for this lab.

## Request Teachers Flow of Events

1. For each lab associated with the course, enter course and lab name, day and time, and weeks during which it will run.
2. Enter skills level required for each person. A lab may be staffed by more than one person. Skills levels are tutor, graduate demonstrator, undergraduate demonstrator.
3. Enter suggested teachers, to help the recruiter.
4. Notify PTT Director.

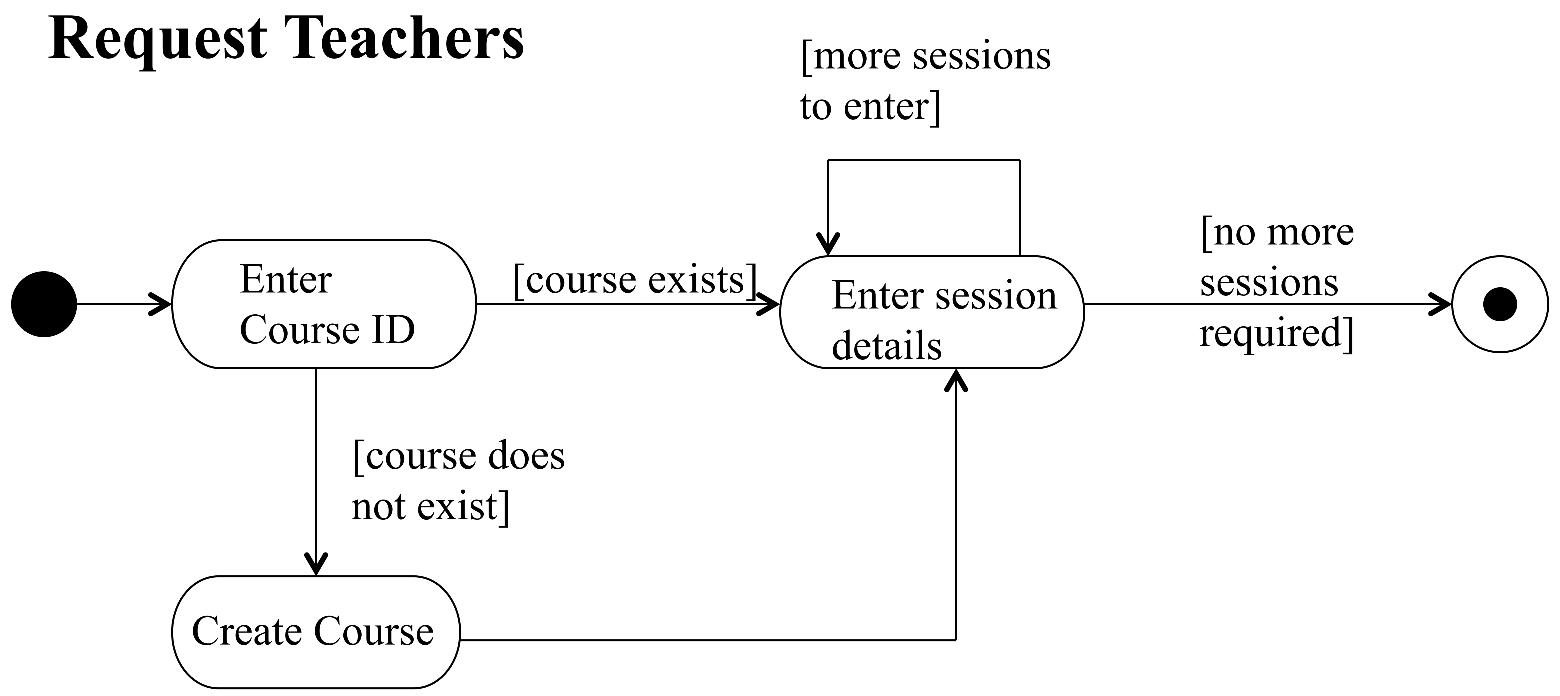
# Alternatives in Numbered List Style

- Alternatives can be indicated using if statements.
  - **if** the course does not exist **then**
  - a) Create the course.
- Repetition can be indicated using for or while statements.
  - **for** each lab
  - a) enter lab name, day and time
  - b) **while** more suggestions to make
  - Suggest a teacher
  - **endwhile**
  - **endfor**
- Don’t get carried away and provide too much detail at this stage.
  - It is not worth trying to code everything during requirements gathering.

# Activity Diagrams

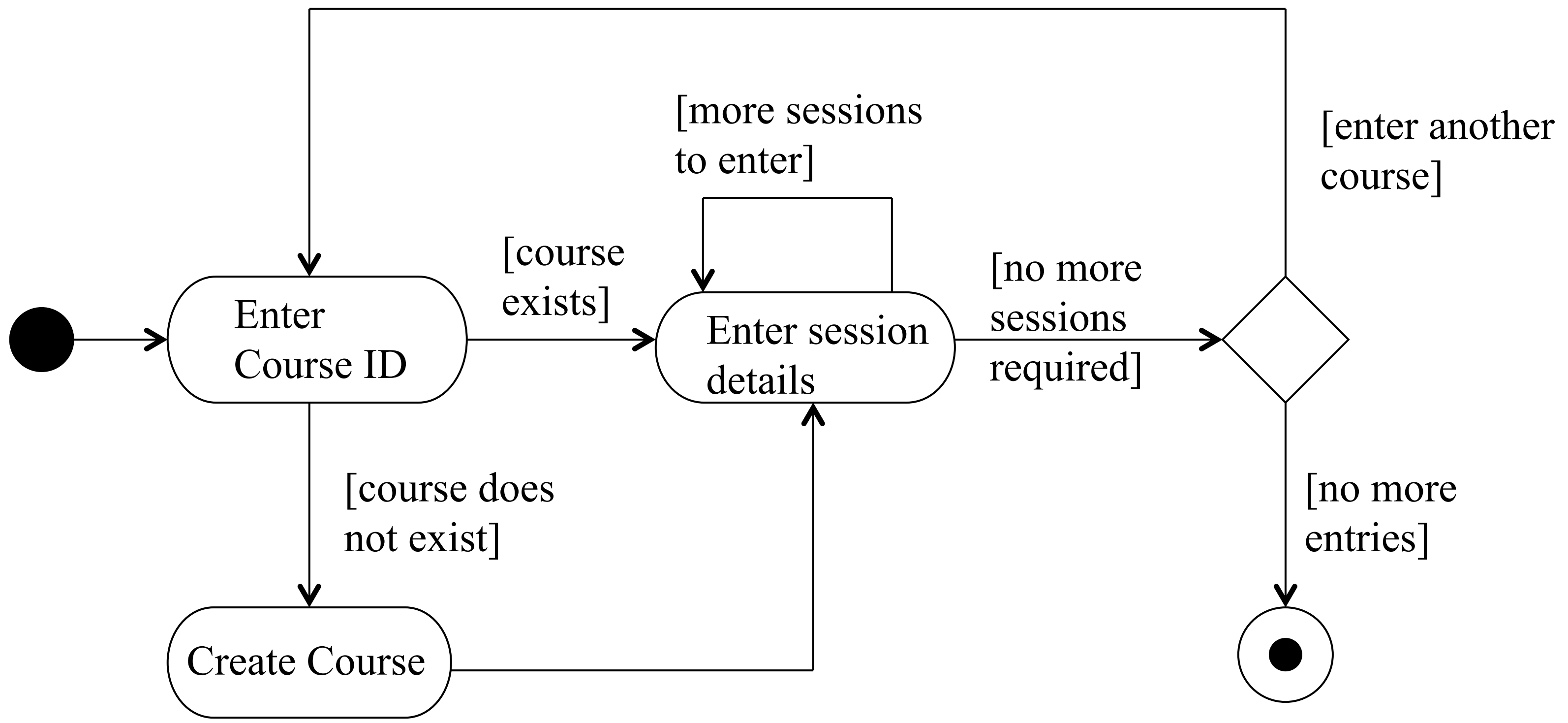
- Activity diagrams are an alternative way of showing the logical structure of the use case details.
- They are useful with more complex use cases, and provide a graphical way of detailing the algorithm.
- Components of an activity diagram are
  - Activities: rounded rectangle
  - Decisions: diamond
  - Guards on decisions: text inside [ ]
  - Fork initiating parallel activity: solid bar
  - Join terminating parallel activity: solid bar
  - Start: solid circle, Stop: donut

# Activity Diagrams (2)



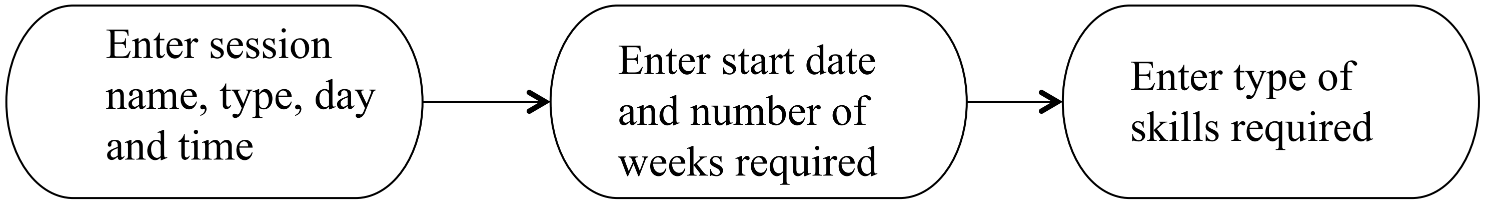
# Activity Diagrams (3)

**Request Teachers (extended)**



# Activity Diagrams (4)

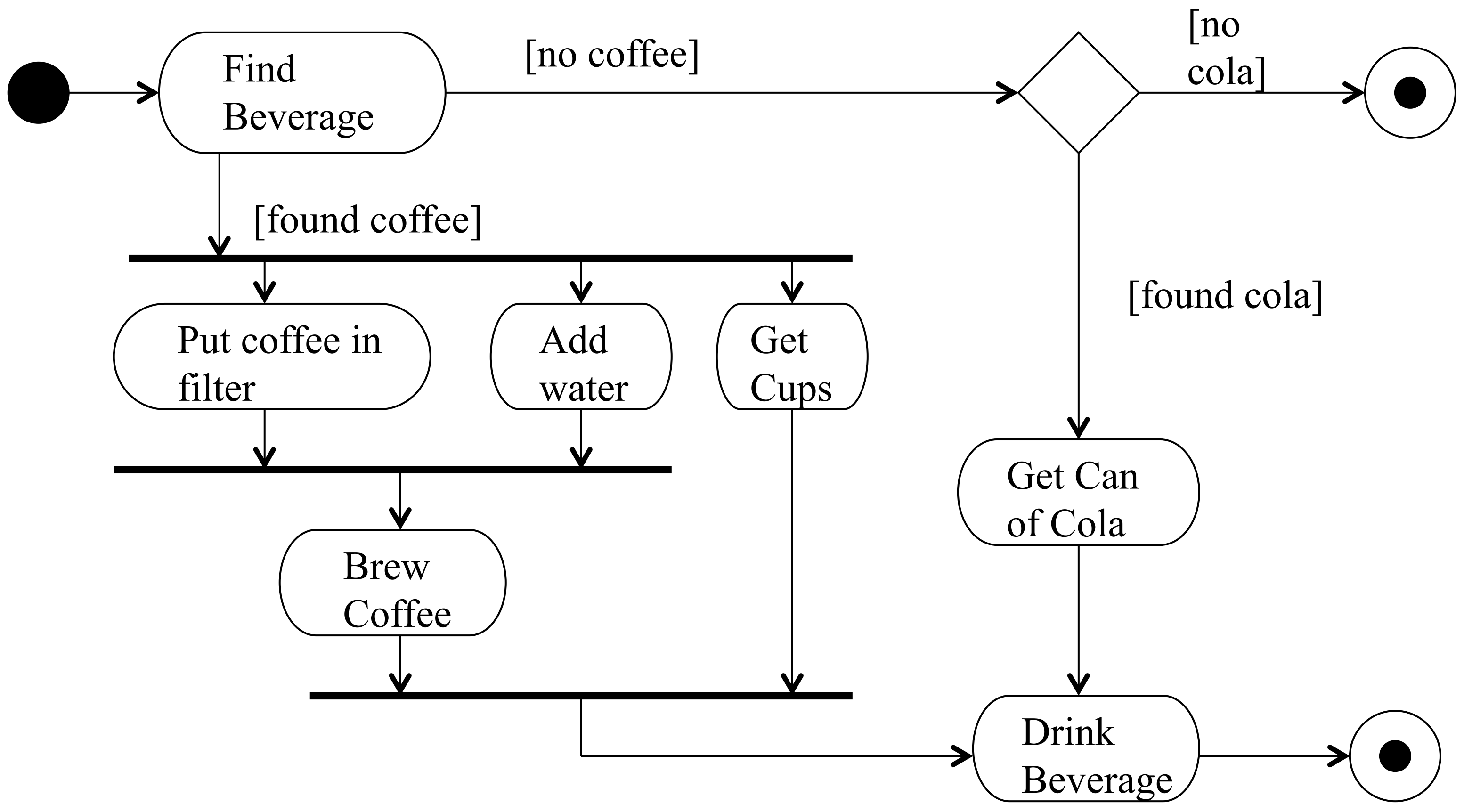
- Good for showing overall flow through use case, especially when testing different scenarios with users, who will be less happy with pseudocode.
- Avoid detailing every action in the activity diagram, **not**:


Enter session name, type, day and time
Enter start date and number of weeks required
Enter type of skills required

## Activity Diagrams – Digression!

- Activity Diagrams are also useful (possible more useful) for showing the overall structure of a system rather than a use case
- During requirements capture use an activity diagram to summarise processes and how they are connected
- Individual processes may then be realised as one or more use cases

# Coffee Break Activity Diagram



## Pre and Post Conditions

- We may occasionally find it useful to record the state of the system before a use case starts, and also the state when it has finished.
- Each use case will have a precondition and a post condition.
- A precondition is a brief description of the state of the system just before the use case starts.
- Similarly, a post condition is the state of the system after the use case has finished.
- If we look at the “Request Teachers” use case, then
  - The precondition is that the course director has logged on to the system
  - The post condition is that a request for teachers has been recorded by our system.

## Rationale

- Each use case should have a rationale.
- This is a free text reason for the use case.
- It is a useful place to record the reasons for this particular use case.
- Also a short English description of what it does.
- The rationale for the request Teachers use case is:

  *The course director records all the labs that he or she plans to run and the staff they would like to employ. This will allow the PTT director to either give permission for the labs to go ahead or else refuse permission.*

  *It is envisaged that informal discussions will have taken place before this request, but entering the request into the system allows the PTT director to use the system to calculate the cost of the labs.*

# Detailed Use Case Description Document

- This is the format recommended for the Use Case Details document that you will hand in.
- One per use case.
- Not all sections are needed for every use case.
  - Just fill in the sections that are applicable.

---------------------------------------------

- Rationale
  - The reasoning behind the need for this use case.
- Actors
  - A list of actors who communicate with this use case, indicating whether they are active or passive.

## Use Case Document (2)

- Priority
  - The importance of this use case to the project.
- Status
  - Where are we in developing this use case.
  - How certain are we that it is correct and complete.
- Preconditions
  - Assumptions made.
- Post conditions
  - The state the system will be left in.
- Extension Points and Included Use Cases
  - See later slides for details.

## Use Case Document (3)

- Flow of Events
  - Either a numbered list or an activity diagram.
  - Both could be included if that would improve clarity.
- User Interface
  - If the use case interacts with people then include a description of the user interface, possibly using story boards.
- Scenarios
  - A list of scenarios used, together with a brief description of what each does.
- Other Requirements
  - Non functional requirements that impact this use case.

# Additional Features of Use Cases

- There are a number of ways we can expand the Use Case Notation:
  - <<include>> allows us to identify common sub-functions shared by several use cases
  - <<extend>> provides a way of adding special variants of a ‘normal’ use case
  - actor inheritance simplifies the Use Case diagram structure when several roles are overlapping

## An Example

**Request Teachers**

```
• Course Director (CD) enters course identifier
• if course does not exist then
          Create a new course
  endif
• for each teaching session required
          CD enters session name, type, day and time
          CD enters start date and number of weeks required
          CD enters type of skills required
          System displays session details
          Suggest teachers for session
  endfor
• Notify PTT Director of needs
```

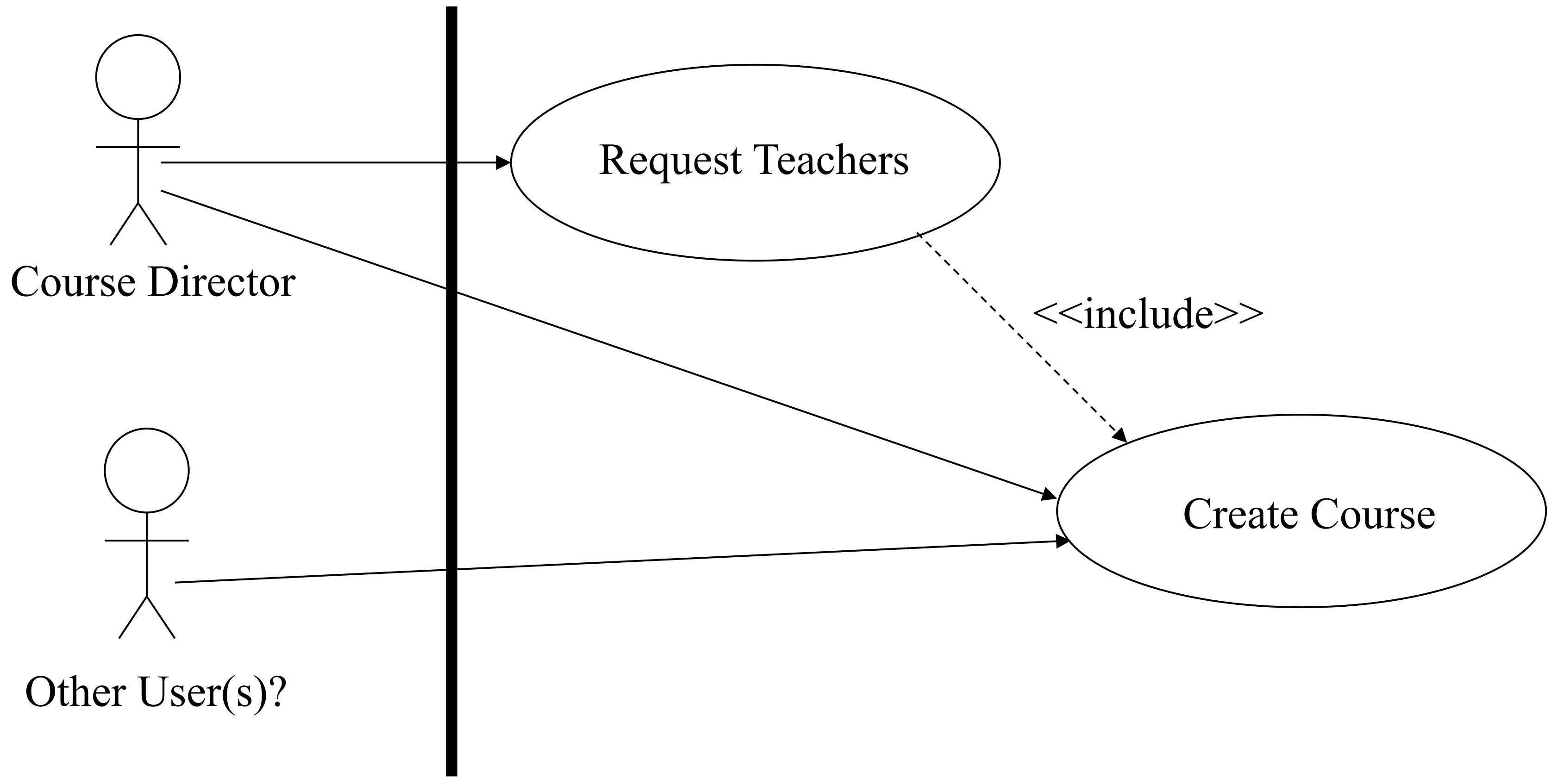
## An Example (2)

- Creating a new course will obviously be needed elsewhere so it makes sense to define a separate use case (Create Course) and include it in Request Teachers.
- Suggesting teachers to fill slots might be considered as a *Would like to have* rather than a *Must have* so could be an extension (Suggest Teachers).

## Include Relationship

- One use case can use another with an <<include>> relationship
- Several different use cases can <<include>> the same use case
- A use case which is included in one or more other use cases can also be used independently

# <<include>> example


Request Teachers
Course Director
<<include>>
Create Course
Other User(s)?

# <<include>> example (2)

**Request Teachers**

```
• Course Director (CD) enters course identifier
• if course does not exist then
        Include Create Course
  endif
• for each teaching session required
        CD enters session name, type, day and time
        CD enters start date and number of weeks required
        CD enters type of skills required
        System displays session details
  endfor
• Notify PTT Director of needs
```

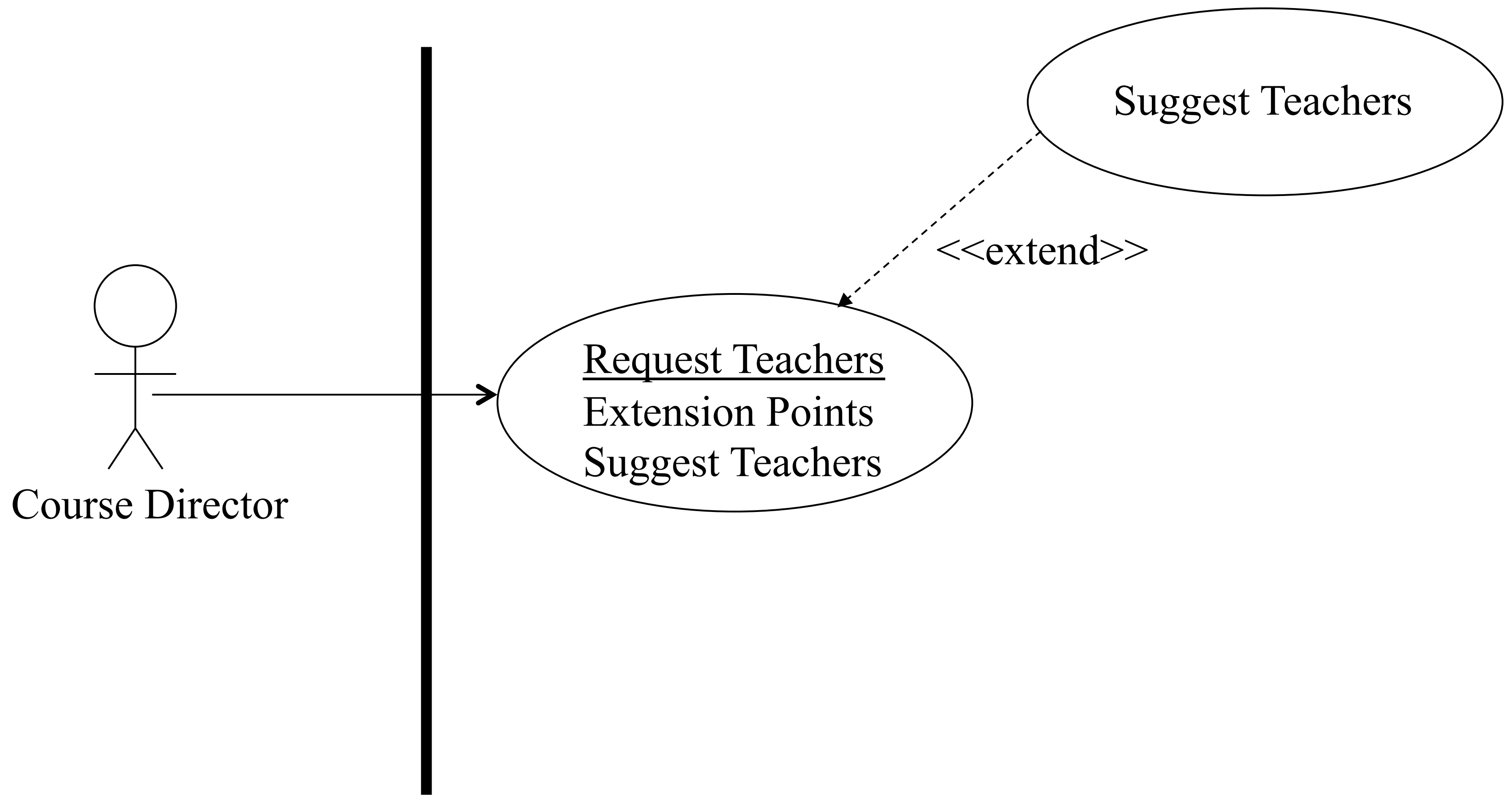
## Extending Use Cases

- One use case will <<extend>> another if we have a branching relationship.
- A common path is initially followed in both use cases, but the flow of control then diverges, to rejoin later on.
- Normally one path is the main stream and the other is the extension.
- The main use case has the common initial steps and also mentions an extension point (or more than one), where branching can take place.
  - The extension point is just a string.

# Extending Use Cases (2)

- The extension use case is connected to the main one by an <<extend>> relationship on the use case diagram.
  - It only mentions the extra steps that take place after the extension use case diverges from the main one.
  - The extension use case details must also start with a conditional statement, the condition for taking the extended path.
  - It must also mention the extension point that it uses.

# <<extend>> example


Suggest Teachers
<<extend>>
Request Teachers
Extension Points
Suggest Teachers
Course Director

## <<extend>> example (2)

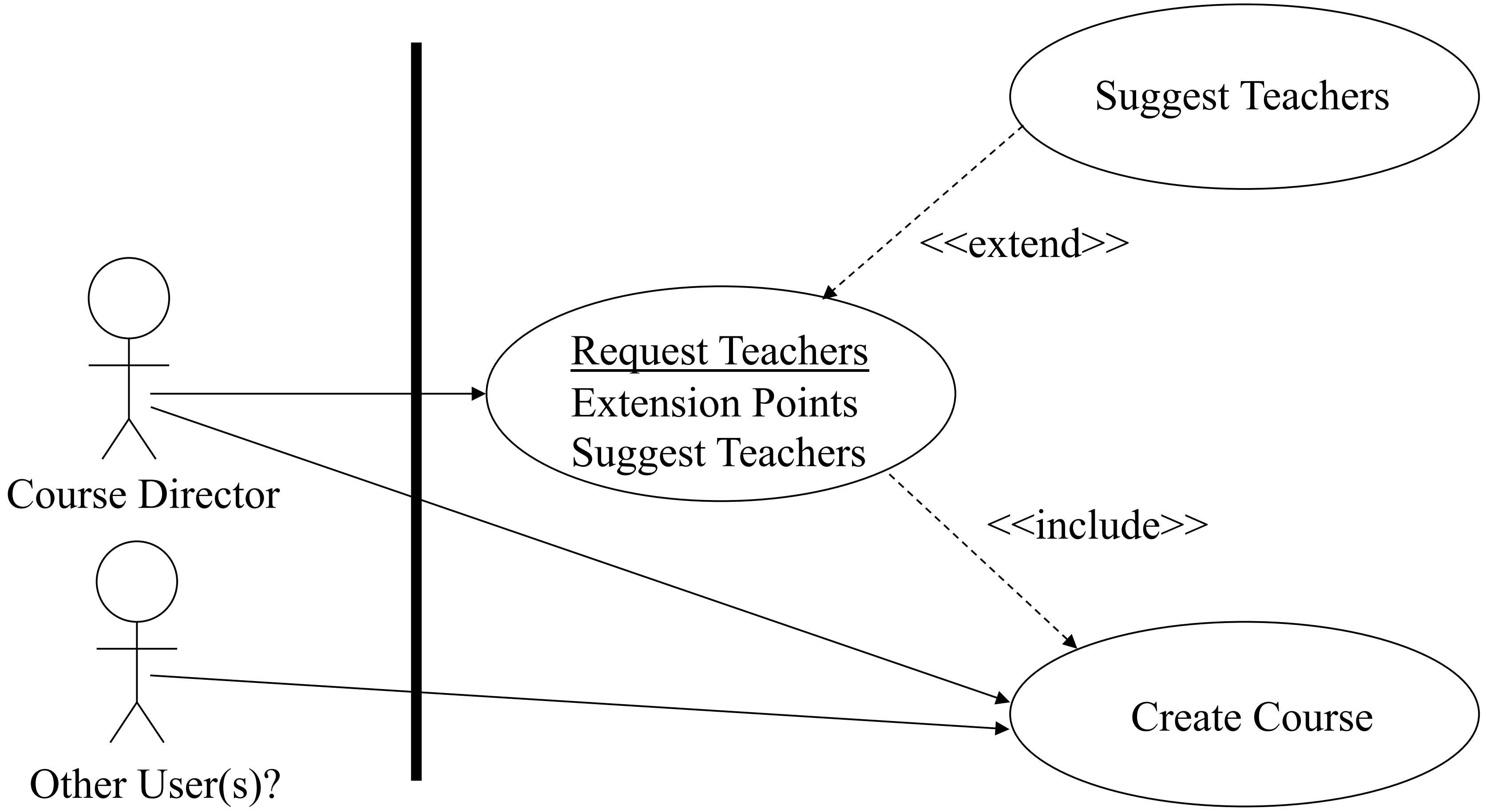
**Request Teachers**

```
• Course Director (CD) enters course identifier
• for each teaching session required
        CD enters session name, type, day and time
        CD enters start date and number of weeks required
        CD enters type of skills required
        System displays session details
        Suggest Teachers Extension Point
  endfor
• Notify PTT Director of needs
```

## <<extend>> example (3)

```
**Suggest Teachers**
if suggestions for teachers then
   at Suggest Teachers Extension Point:
while more suggestions for teachers to be made
        CD suggests name of teacher for session
endwhile
```

# Combining <<include>> and <<extend>>



## Combining <<include>> and <<extend>> (2)

**Request Teachers**

```
• Course Director (CD) enters course identifier
• if course does not exist then
        Include Create Course
  endif
• for each teaching session required
        CD enters session name, type, day and time
        CD enters start date and number of weeks required
        CD enters type of skills required
        System displays session details
        Suggest Teachers Extension Point
  endfor
• Notify PTT Director of needs
```

## <<extend>> vs <<include>>

- <<extend>> permits the addition of special actions to a use case; the original use case is complete without any extensions.
  - an extension may be treated as lower priority than the original use case ('would like to have')
- <<include>> is a way of sharing functionality; any use case which includes another is incomplete without the included use case, which imposes a priority order
  - a use case which is included in another can be used independently of any use case that includes it.
  - <<include>> was called <<uses>> in earlier versions of UML
- Do not use <<extend>> or <<include>> to break the problem down into very fine detail

## Another Example: Place Order Use Case

1. Customer selects place order
2. Customer enters name and address
3. Customer enters product codes for products ordered
4. System supplies product descriptions and prices

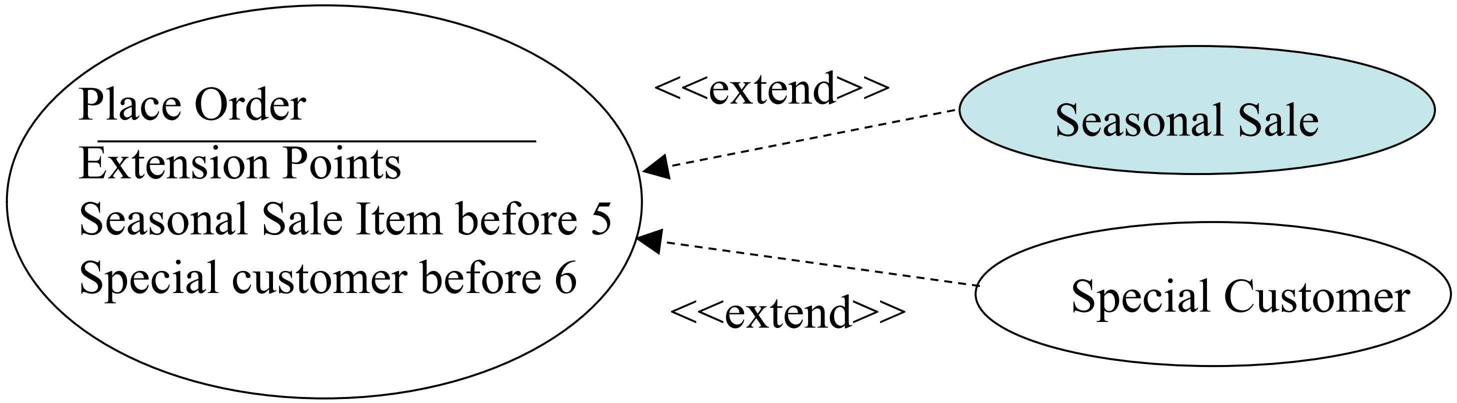
   Seasonal Sale Item Extension Point.
5. System keeps a running total of total price.

   Special Customer Extension Point
6. Customer enters credit card information.
7. Customer selects submit
8. System verifies the order; saves it as pending; and notifies accounts.
9. When payment is confirmed, the order is marked as confirmed and an order id is sent to the customer.

# <<extend>>: Seasonal Sale

If product in sale list then At Seasonal Sale Item Extension:

1. System will get the sale discount for the product
2. System displays the discount on this product
3. System calculates a discount for the order line
4. System subtracts the discount amount from the item cost

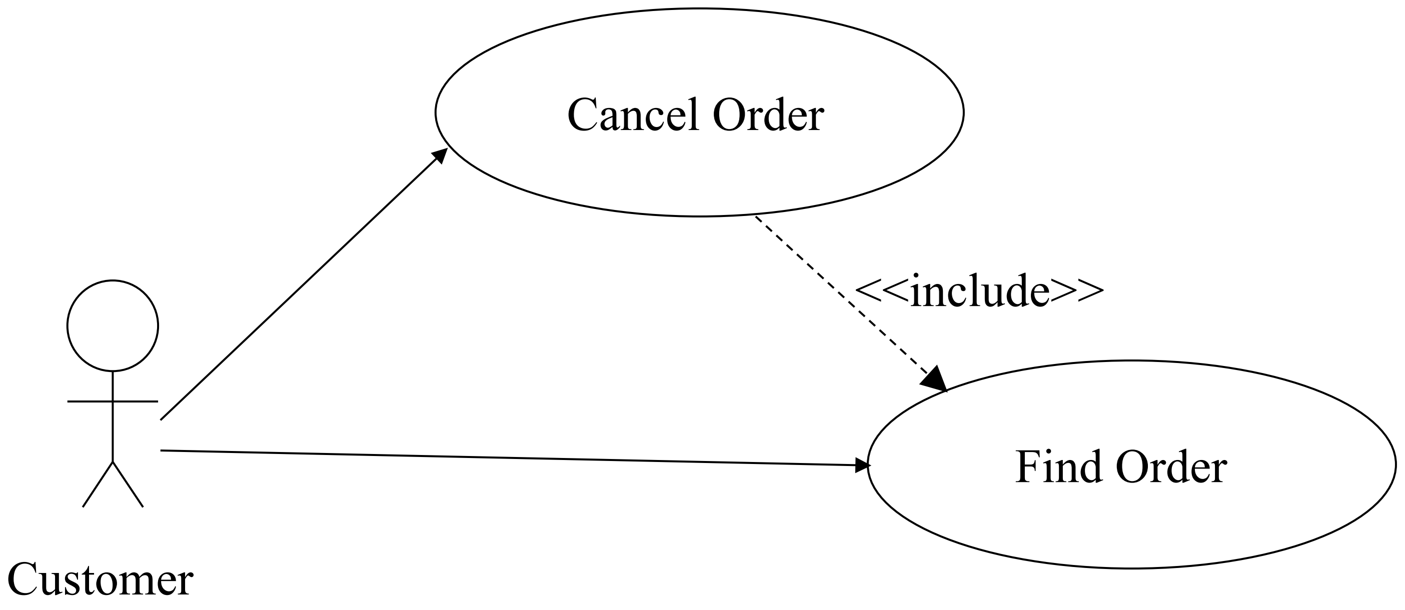


## Another Example: Cancel Order

**Cancel Order**

1. The customer requests cancelling an order
2. Include Find Order
3. If the order status is confirmed
    a. The order is marked cancelled.
    b. Accounts is notified
4. If the order has been shipped
    a. The customer is notified of our return policy.

# <<include>> Find Order

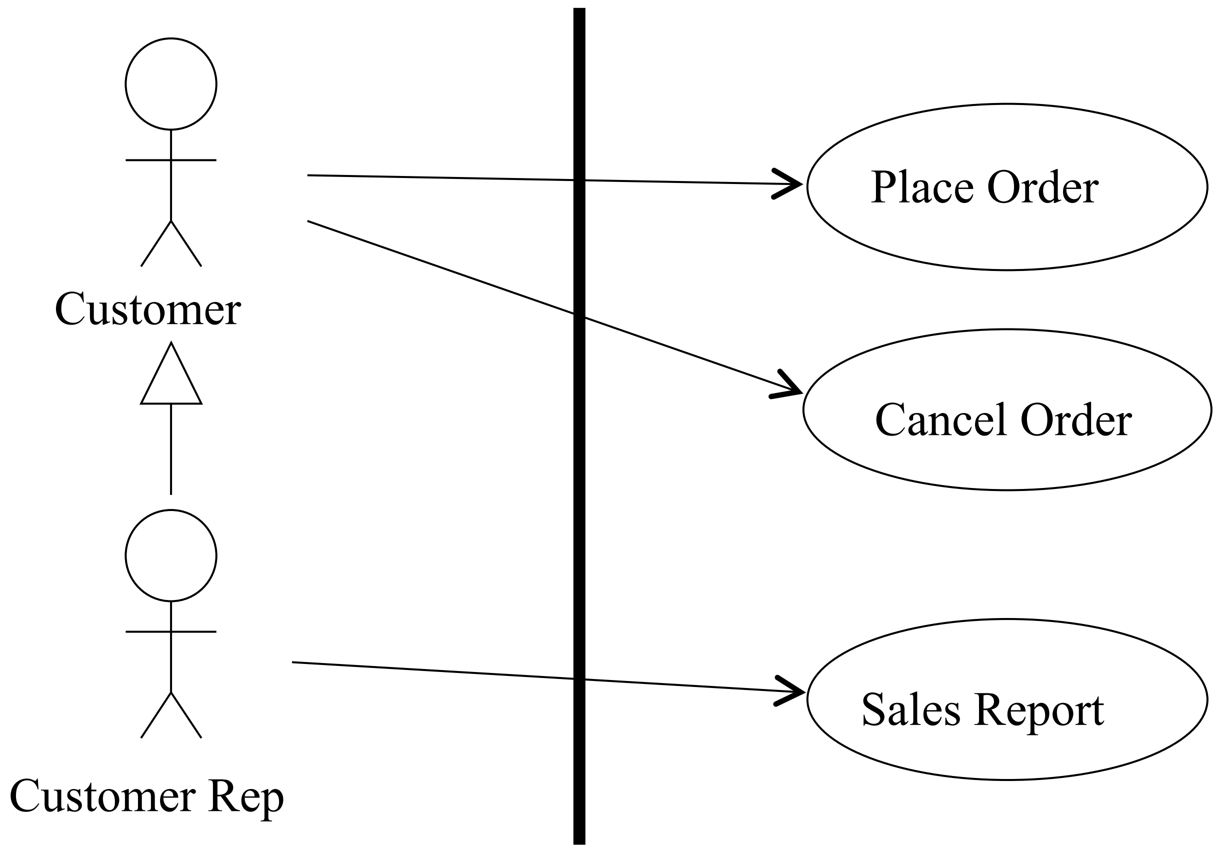


The Customer can  also use Find Order directly to check the status of an Order.

## Actor Inheritance

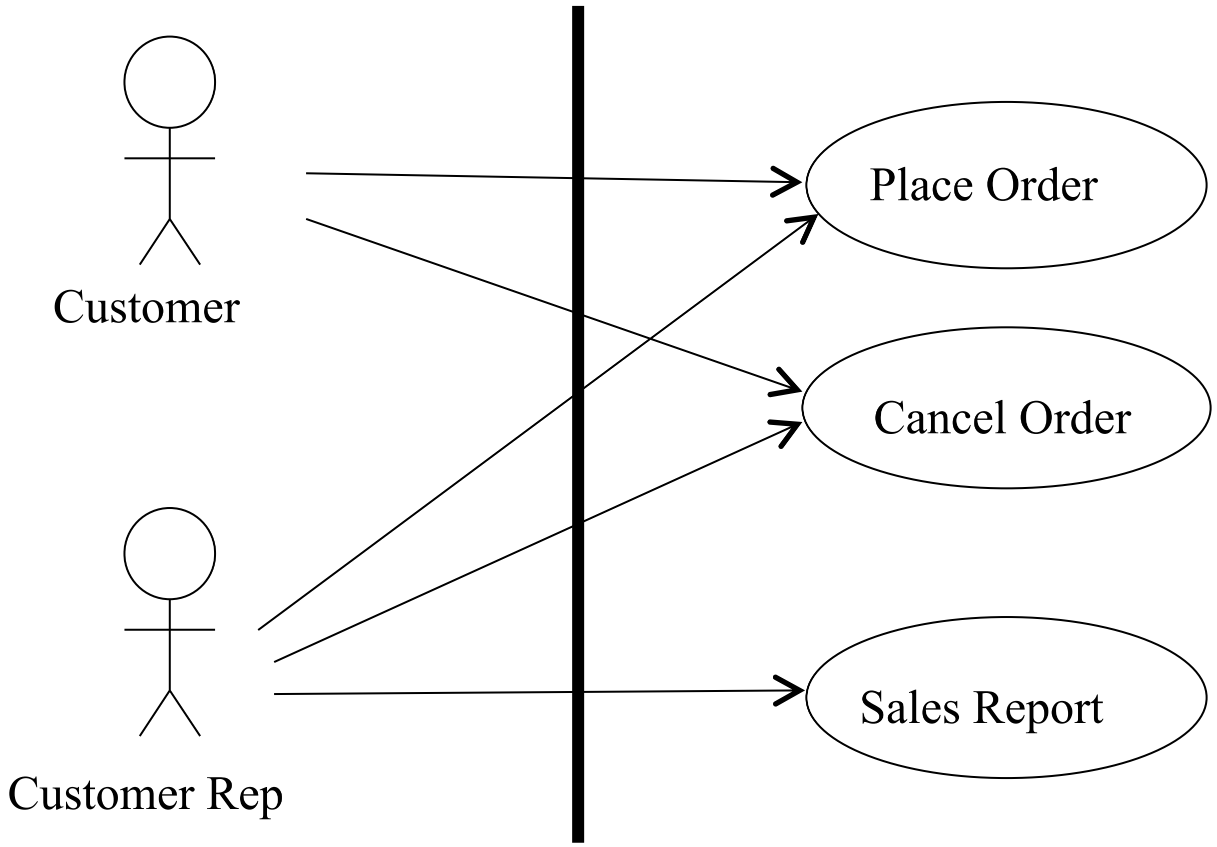
- It is possible for two actors to require very similar functions, but have significant differences as well.
- This shows up on the use case diagram.
- We can simplify the diagram by saying that one actor inherits from another actor.
- This is represented by the arrow connecting Customer Rep to Customer.  A Customer Rep has access to the same use cases as a  Customer plus Sales report.

# Actor Inheritance (2)



# Actor Inheritance (3)

*This has the same meaning as the previous slide*



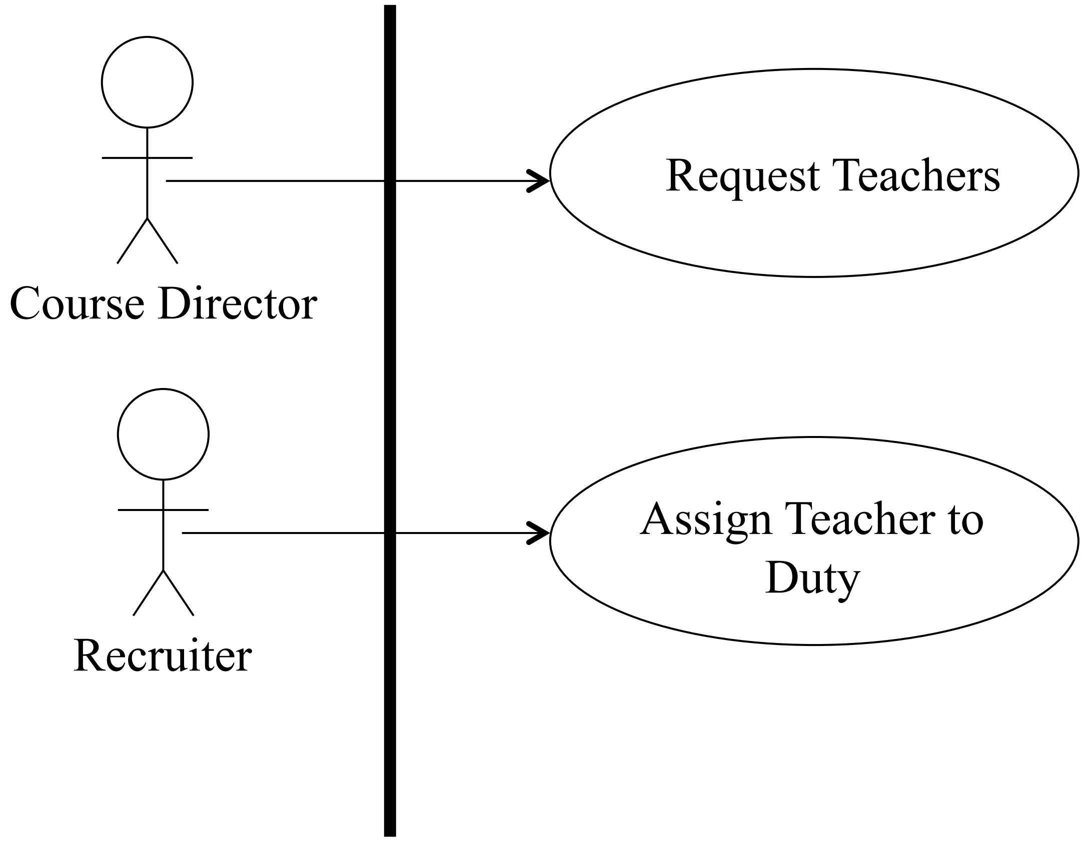
## Use Cases - summary

- Show **functionality** of a system - user view of system
- Define **scope** of system, what is inside the system boundary (or what is excluded)
- Define the **actors** that will interact with the system, initiate actions, receive information, etc. (Also interactions with external systems)

# Use Cases - summary (2)

- Do **not** show data structures - there are often hidden connections (dependencies) between use cases through the stored data. We need **classes** (a domain model) to define the object-oriented data structures.
- They are not very good for capturing non-functional requirements, which are essentially documented separately.
- They do not show how the system is embedded in its **environment** - there are often important activities happening externally to the system; e.g. in our PTT example, the claim/ payment activities are mainly external to our system.
  - This is a common problem with most system models.

## Use Cases Summary (3)



*Note: Data storage is not shown*

## Use Cases - summary (4)

- In the Unified Process, use cases ‘drive’ software development.
- Capturing user requirements; use cases help to structure the requirements and capture details
- Iterations - each iteration of software development constructs the software to support a subset of the use cases, prioritised by risk and how essential they are.
- Testing - use cases should drive the testing of the systems they define the user requirements
  - Each scenario associated with a use case should generate at least one test case