Use Cases

Introduction

Actors and Use Cases

- The ‘Use Case’ approach to developing software came before the current Agile approach.
  - More documentation was required.
  - More time was spent analysing and designing the system before starting to code.
- A Use Case is the equivalent of a User Story.
- An Actor is the equivalent of a User Role.
- Several pages of documentation are required for each user story.
- All the Actors and Use Cases are combined in a Use Case Diagram to give the Big Picture.

Why Learn about Use Cases?

- The approach is old-fashioned and not widely used any more.
  - Because of the excessive documentation.
- Some techniques are still useful and can add to an Agile approach.
  - In some cases there is too little design and documentation in Agile approaches.
- The Unified Modelling Language (UML) is still widely used, even in Agile processes.
  - It has a number of useful diagrams to support Use Cases.
  - They can be generalised to User Stories.
- I will use the PTT example to illustrate the Use Case approach.
  - Don’t worry if your solution to the Assessed Exercise is different.

The Project Description

- The project starts with a text document describing what the system will do.
- It is then used to generate Use Cases.
- The Use Cases are designed and only then can they be implemented.

Identifying the System Boundary

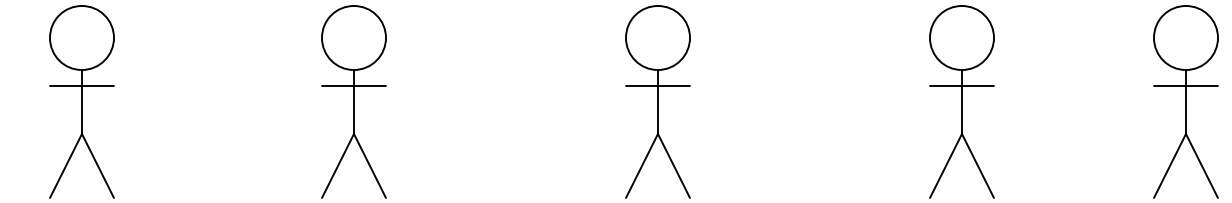
- Once we have an initial idea of what our system will do, we need to identify the system boundary and those things that lie inside and outside it.
- If something lies inside the boundary then we have to create it.
- If something lies outside the boundary then it is already there, we do not have to worry about creating it.
  - We just have to interact with it.
- We discover what the boundaries are by describing the actors and the use cases.

Actor Roles

- Objects outside the system are called actors. They initiate activities by using our system (active actors) or respond to activities (passive actors).
- Actors are anything that interfaces with our system.
  - They can be people, other pieces of software, hardware, databases or networks.
- We are interested in actor roles rather than individual actors.
  - One person might take on several different roles, and hence be represented by several different actors.
  - Conversely, several different people may all perform the same role and be represented by a single actor.

Actors

- There will usually be several class directors, one for each class, but they will be represented by just one actor role, since all class directors interface with our system in the same way.
- One person may recruit people and also pay them. This person will be represented by two different roles.
- Actors are traditionally represented by stick figures with the name of their roles underneath.
- We can think of the following actors for our system.



Actors (2)

- Each actor needs a short description defining their role in the system.

  Class Director*: A person who requires part time teachers for his class.*

  Recruiter*: Finds people to do the teaching.*

  Teacher*: A part time teacher.*

  Payer*: Organises payment for teachers.*

  Trainer*: Trains part time teaching staff.*

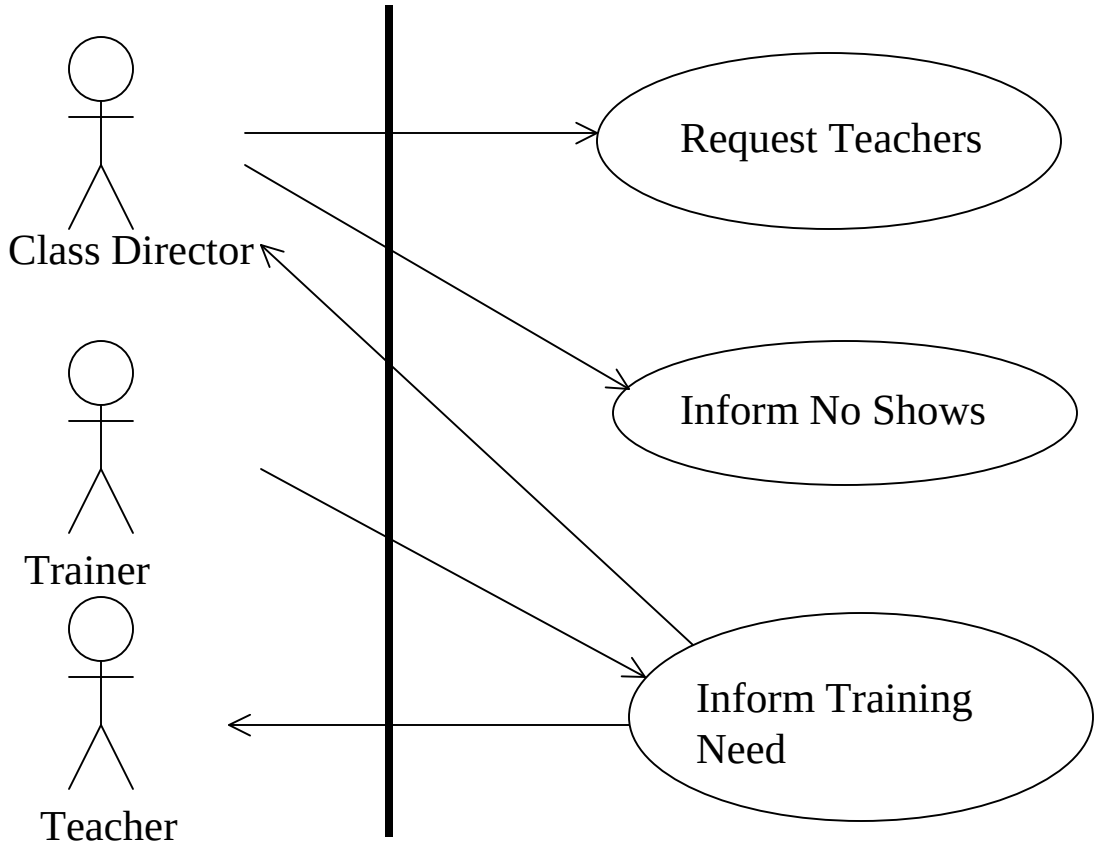
Actors (3)

- Identifying the actors helps us to define the system boundary.
- Our system is not responsible for training the teaching staff, just for passing them on to the appropriate training organisation.
  - The trainer actor
  - We maintain a list of skills and training needs for each teacher.
- Similarly, we do not pay them ourselves.
  - The payer actor.
  - We maintain a list of payments made and payments due, together with our budget.

Use Cases

- Use cases define the insides of the system.
- We find the use cases by looking at the actors and seeing what activity they initiate.
- We record the use cases as ovals with a simple description.
  - This is the UML notation which we will use for this course.
  - I will extend the notation slightly in several places.
- Use cases should be short, doing one thing well, since it is less likely that they will be ambiguous.
- Let us look at three of the use cases associated with the class director and the trainer.

Use Cases (2)


Request Teachers
Class Director
Inform No Shows
Trainer
Inform Training Need
Teacher

Use Cases (3)

- I have used arrow on the actor lines to indicate whether an actor is active or passive.
  - This will be different for each use case.
  - Class director is active for some use cases and passive for others.
- Lines go from the active actor to the use case, and then from the use case to any passive actors.
- In many cases we do not show the lines to passive actors to avoid clutter.
- The arrows can also be left out.

Refining the Project Description

- These use cases have helped us to think about what the system will do, and what it will not do.
  - In this example, we have decided that our system will not handle training of part time teachers, but delegate this work to the university and class directors.
  - This leads to an additional passive actor, the university system for training part time teachers.
- Another point of interest is that we do not record all of the activities of the actors.
  - For example, the class director also organises courses, but we do not record this information because it is not part of this system.

Active and Passive Actors

- Active actors will generate the use cases.
- If the actor is a person, think of him or her sitting down at a computer and using a program.
- In our example, the class director is an active actor when requesting teachers for his classes.
- A passive actor is contacted by the use case, and does not initiate any activity.
- In our example, the class director is a passive actor in the “Inform class director of training need” use case.
- Think of him receiving e-mail (generated by the program) with a list of people, together with the training that he has to initiate.
  - The actual implementation might be different.

Timed Activities

- Sometimes our system will generate activities internally.
- The use case will involve actors outside the system, but they are all passive actors.
- The most common way of doing this is by timed activities.
  - Some basic system functionality is scheduled to run later.
  - Teaching staff have to submit claims forms before the second Monday of a month in order to get paid at the end of the month.
  - We might send out automatic reminders.
- A common way of dealing with this is to have an internal TIMER actor which generates use cases.

TIMER

Boundary Problems

- Here are some common problems that arise when defining the system boundary.
- Are some of the requirements being handled by an actor?
  - Actors are outside our system and so cannot actually handle requirements. They will generate requirements.
  - We must redefine the actor role and generate new use cases to handle these requirements.

New Requirements

- What if we discover new requirements during the process of identifying actors and use cases?
- We should ask the following questions.
  - Are these requirements necessary for the system?
  - Are they something our system would naturally do?
  - Can they be handled by one of the existing actors?
  - How do they affect our current risk analysis?
- If we have added several new requirements we may have to step back and look at the complete system again.
  - Function creep, gold plating or discovery of essential functionality?

The Scope of the Project

- The process of accumulating actors and use cases goes a long way towards defining the scope of the project.
- After a time we must step back and see what we have got.
- Are the system requirements all met by use cases?
- Examine the project description to see if we have covered everything.
- If there is something missing, then we need to go back and find some more use cases or actors.
- Alternatively, the use case approach may have generated more requirements, which can be added to the project description.

Non-Functional Requirements

- Some of the requirements cannot be met by defining use cases.
  - Use cases are functional requirements and define the functionality of the system.
- Other requirements are called non-functional requirements.
- Typical examples are performance or security requirements.
- These non functional requirements can apply to the whole system or to an individual use case.

Non Functional Requirements (2)

- For example, the class director must give at least one weeks notice of teaching requirements, to allow for recruitment.
  - This is a non-functional requirement for the “Request Teachers” use case.
- The information stored must always be up to date and consistent at the end of each working day.
  - A system wide non-functional requirement.

Prioritising Uses Cases

- Projects have start and finish date, with fixed budgets.
- It is often not possible to do everything that we want to, at least not immediately.
- We must prioritise the use cases to make sure that the important things get done and no effort is wasted on unimportant features.
- A typical priority scheme will have the following categories
  - Must have
  - Should have
  - Could have
  - Would like to have
- This is called the MoSCoW approach.

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 into more detail for each use case.
- We will learn more about the system functionality and be in a position to start designing the system.
- The most important use case details that we want to establish are the flow of events.
  - This is like programming but in a program-design language.

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 scenarios, 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 and list all possible scenarios for each use case.
- The equivalent of scenarios in an Agile approach are different story tests.

Alternatives

- A scenario cannot have any if 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 class 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 the secondary scenarios.

Request Teachers Secondary Scenarios

- Class does not exist
  - Option to create the class
- Class does not have a lab
  - Option to create the lab
- Suggested teacher not in the system.
  - Option to add teacher to the system

Flow of Events

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

Request Teachers Primary Scenario

- Rob accesses the system and brings up class 1.
- He enters the lab for Tuesday 10-12 in semester 1.
- He enters the need for a tutor and an undergraduate demonstrator.
- He enters the Lab for Wednesday 3-5 in semester 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 class, enter class 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.

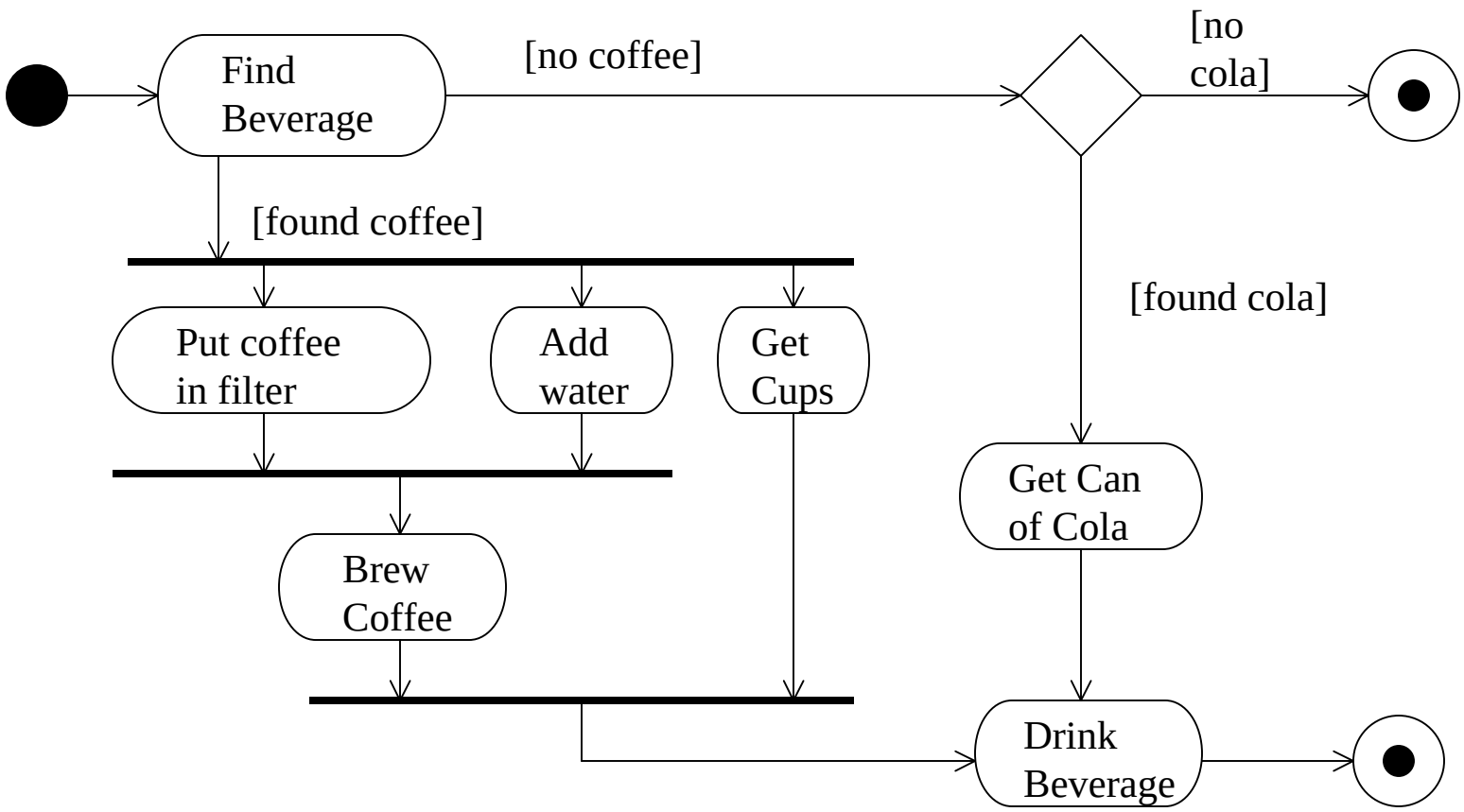
Alternatives Using the Numbered List Style

- Alternatives can be indicated using if statements.
  - If the class does not exist then
  - a) Create the class.
- 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
  - end
  - end
- Don’t get carried away and provide too much detail at this stage.
  - It is not worth trying to code everything during requirements gathering.

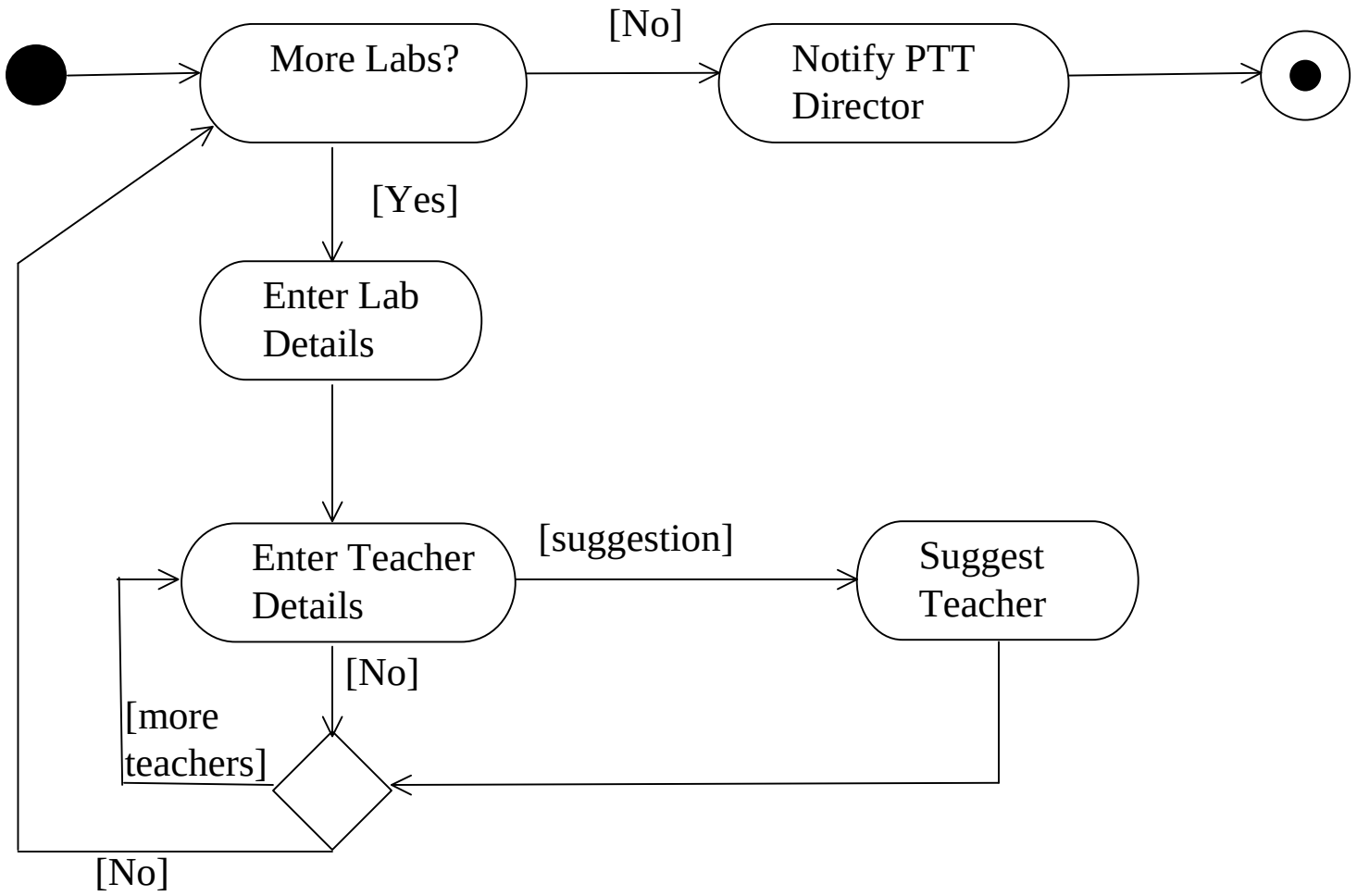
Activity Diagrams

- 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.
- They are not good at showing repetition.
- Components of an activity diagram are
  - Activities: rounded rectangle
  - Decisions: diamond
  - Guards on decisions (if statements): text inside []
  - Fork initiating parallel activity: solid bar
  - Join terminating parallel activity: solid bar
  - Start: solid circle, Stop: donut

Coffee Break Activity Diagram



Request Teachers Activity Diagram



Rationale

- Each use case should have a rationale, the reason for the use case.
- Also a short English description of what it does.
- The rational for the Request Teachers use case is:

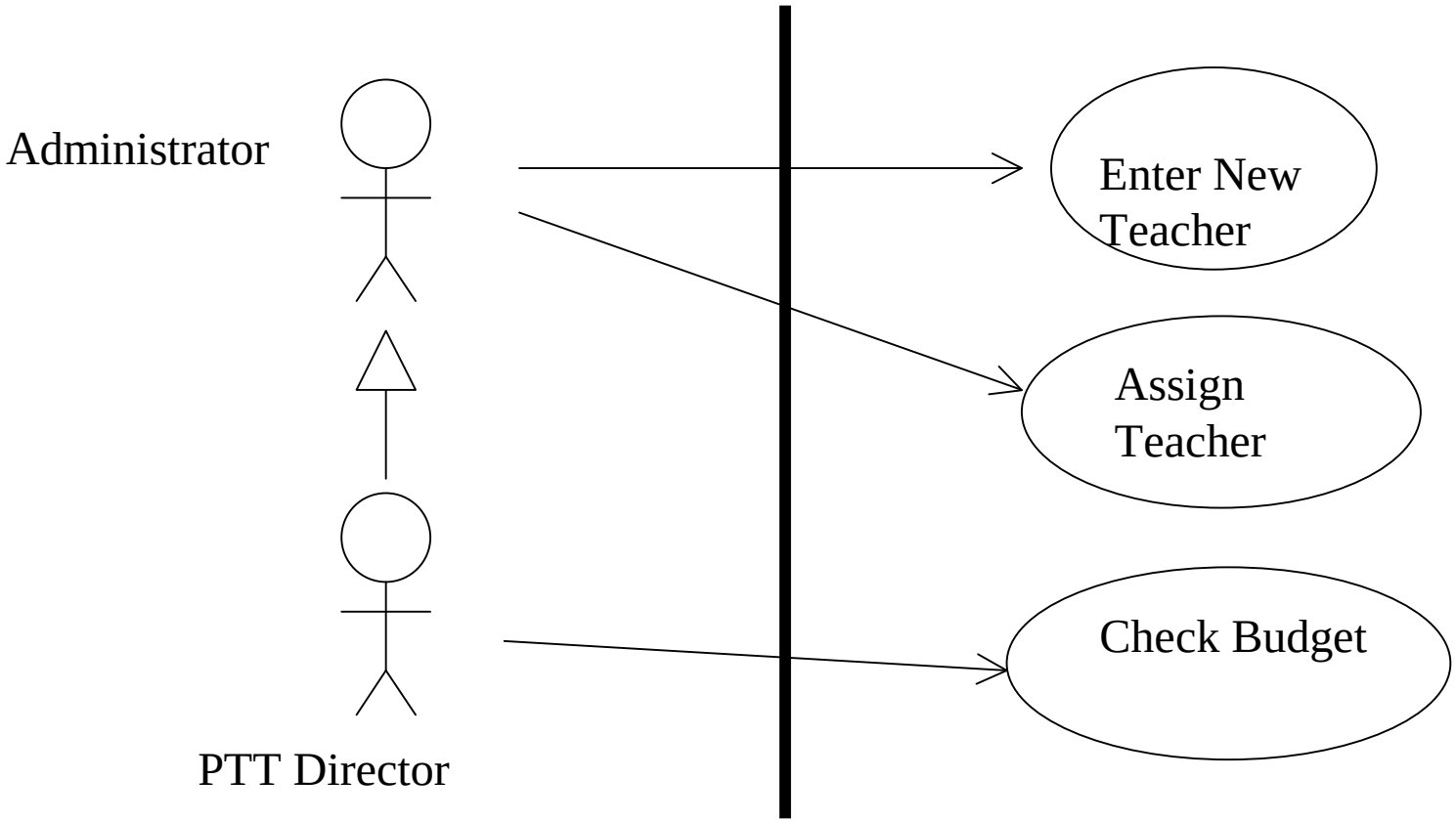
  The class director records all the labs that he plans to run and the staff he would like to employ. This will allow the PTT director to either give permission for the labs to go ahead or else refuse permission.

  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.

Actor Inheritance

- It is possible for two actors to be very similar, 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.
- We can add another actor: Administrator.
  - PTT Director and Payer both inherit from Administrator.
  - They are both special cases of Administrator.
  - They are both an Administrator + extra functionality.
- This is represented by arrows connecting PTT Director and Payer to Administrator.

Actor Inheritance (2)



Questions

1. Read the project description on the next slide and identify the actors and use cases. Draw a use case diagram showing how they interact.
2. Write down use case details for three of the use cases that you discovered. You will have to take on the roles of the subject matter experts again and make sensible assumptions when working out the details.

Questions

Newsagents plan to use a highly sophisticated computer system to make sure that the papers get delivered each day. The system should be able to cope with the needs of both the paperboy (or girl) and the newsagent. Each morning the paper person receives a list of all the houses on their run, together with the papers and magazines that they must deliver. They also gets a list of totals for each publication so that they can just collect the required totals of each of the various papers and then make up their run. The newsagent will be able to enter new orders for papers, record complaints and temporary changes, and be able to calculate the pay, based on the going rate, minus deductions based on mistakes. The newsagent will also be able to take orders for a large range of magazines and calculate daily totals for all the different publications.