

- What is a use case?
- Why use cases
 - requirements and specification?
- Properties of use cases.
 - Actors
- Use case Diagrams.
- Some brief examples
- Next time the use case description

What is a Use Case?

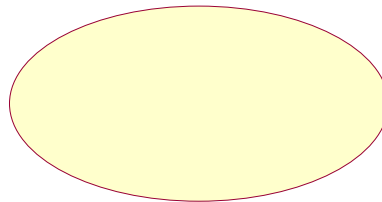
- A use case is a modelling technique used to describe **what** a new system should do or what an existing system already does from the user's point of view. [Note: Michael Jackson's specification interface (1995)]
 - An (important) aim of use case modelling is to describe the functional requirements of the system.
 - The functionality of the system is represented by a complete set of *use cases*.
 - Each use case specifies a “complete functionality”: one general usage of the system.
- *We might argue to what extent they specify*

A definition

- A use case specifies the behaviour of a system (or a part of the system):
- it is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an *actor*.
 - Booch et.al (1999)

What does it do?

- A use case describes the sequence of events of an actor (an external agent) using a system to complete a process.
- They are stories (scenarios) or cases of using a system.
- The UML notation is :-



Buy Items

Why Use Case modelling?

- A use case provides an *efficient* communication mechanism between end-users and developers
- Represents what the system offers
- It defines a testable system requirement from an outside-in perspective.
- *Note: This is the party line....*

- Provide structure:
 - Identify the actors
 - For each actor
 - find what they need from the system (which use cases have value for them)
 - Find any interactions they expect to have with the system (which use cases they take part in for others benefit)
- Provide priorities:
 - How much an actor needs a given use case

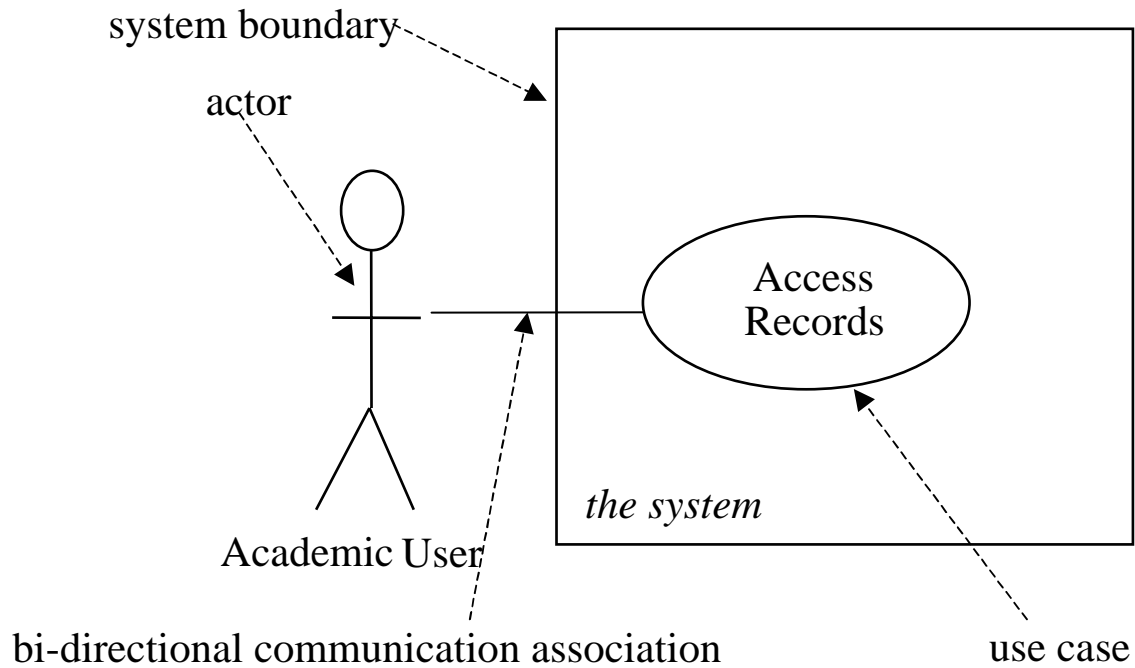
- According to the UML User Guide
 - Establish system context by identifying the actors that surround it.
 - Consider the behaviour that each (actor) expects or requires
 - Name these common behaviours as use cases.
 - Factor common behaviour into new use cases that are used by others;
 - factor variant behaviour into new use cases that extend more main line flows.
 - Model these use cases, actors, and their relationships in a use case diagram.
 - Adorn use cases with notes that assert non-functional requirements;
 - you may have to attach some of these to the whole system.

- Analysis of the system and the development of a use case model means that you should have the following aids to planning
 - have a good idea of what each use case means
 - an understanding of who wants each and how much
 - knowledge of which use cases carry most risk
 - a plan for how long it should take to implement each use case

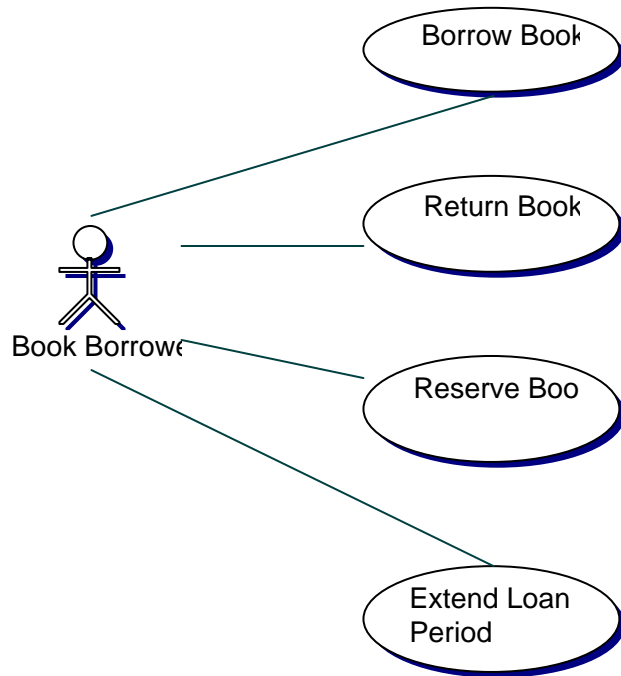
- Each use case describes a requirement on the system, so a correct design will mean that each use case is realised.
 - Therefore to validate a design - check that each use case can be carried out.
- Use cases provide enough information to derive systems tests and validation tests.

- In any system, some things are inside the system and some are external. Those things inside the system are responsible for carrying out the behaviour that those outside the system expect the system to provide.
- Those things external to, but interacting with the system, constitute the context. The context defines the environment in which that system lives.
- *In theory (for the purists) we do not concern ourselves with internal system behaviours.*

Use Case Diagram

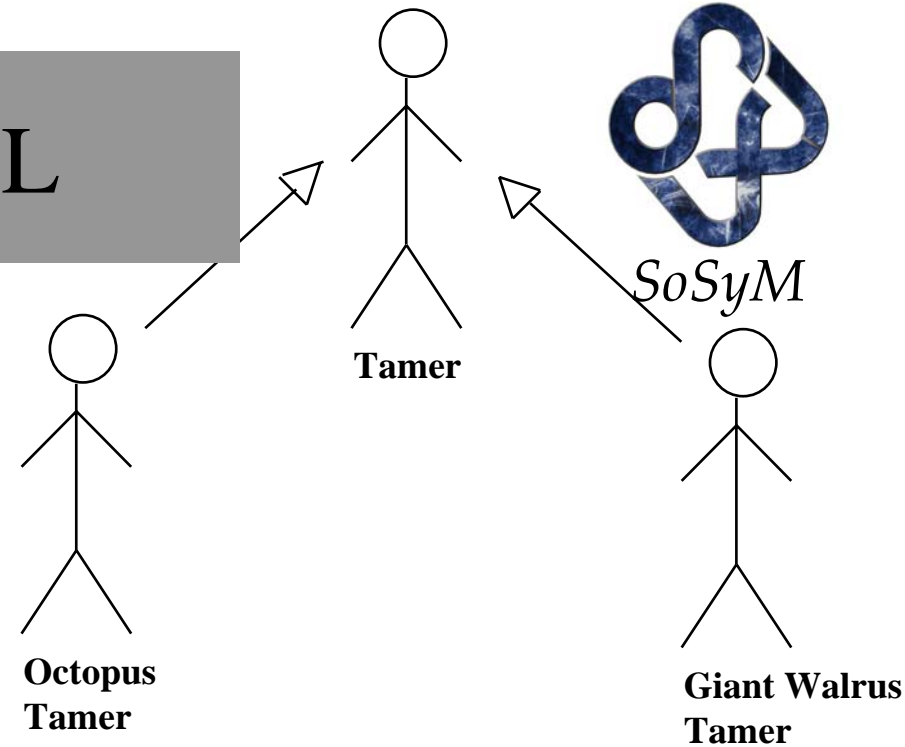


Another Use Case Diagram



- Cross references Actors to Use Cases
- An actor is a stick person
- A Use Case is an ellipse
- Lines represent interactions between the actors and Use Cases

Actors In UML



The UML User Guide defines *actor* as:

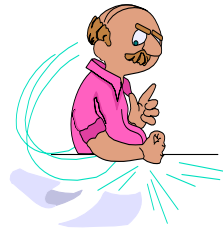
“A coherent set of roles that users of use cases play when interacting with use cases.”

That means we can show inheritance and treat *actor* as a gen-spec structure. Hence, an actor when instantiating a use case is in fact playing a *role*.

What is an actor?

- **Anything with which the system interacts**

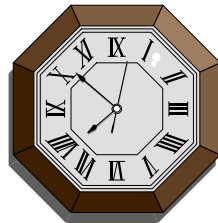
- a human



- another system

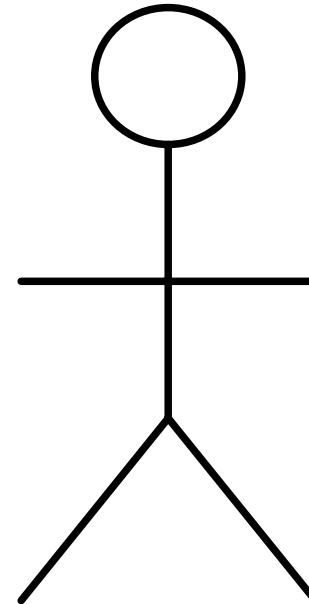
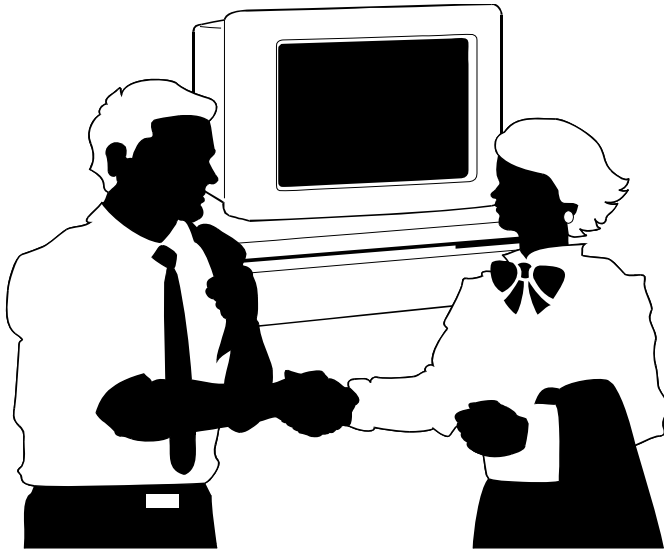


- time



- Actors
 - exchange information with the system
 - directly interact with the system *without the intervention of another actor* [again a bit purist]
 - are external to the system
 - perform non-deterministic actions (*though dependencies can be considered - see later*).
 - they are responsible for carrying out the use cases they are associated with
- Actors should not be described or analysed in detail (*will also discuss later*).

Users, roles, actors and resources



Identifying an actor -1

- An actor may:
 - Only input information to the system
 - Only receive information from the system
 - Input and receive information to and from the system
- Potential human users are relatively easy to identify and will be a first cut solution.
- Subsequent iterations will identify the roles
 - e.g. In the library you identify a student and a lecturer as potential actors. From the library systems point of view they both do the same thing i.e. borrow a book. The role is that of BookBorrower and that is the actor.

Identifying an actor - 2

- A non-human actor can be more difficult to identify
- A keyboard is not an Actor because it is operated by a human.
 - Input from a barcode reader?
 - Input from another system in the same company? (the Internet?)
 - The output goes to something similar?
- If it is clear what an external device or system is, there are different views about whether they should be shown on Use Case diagrams.

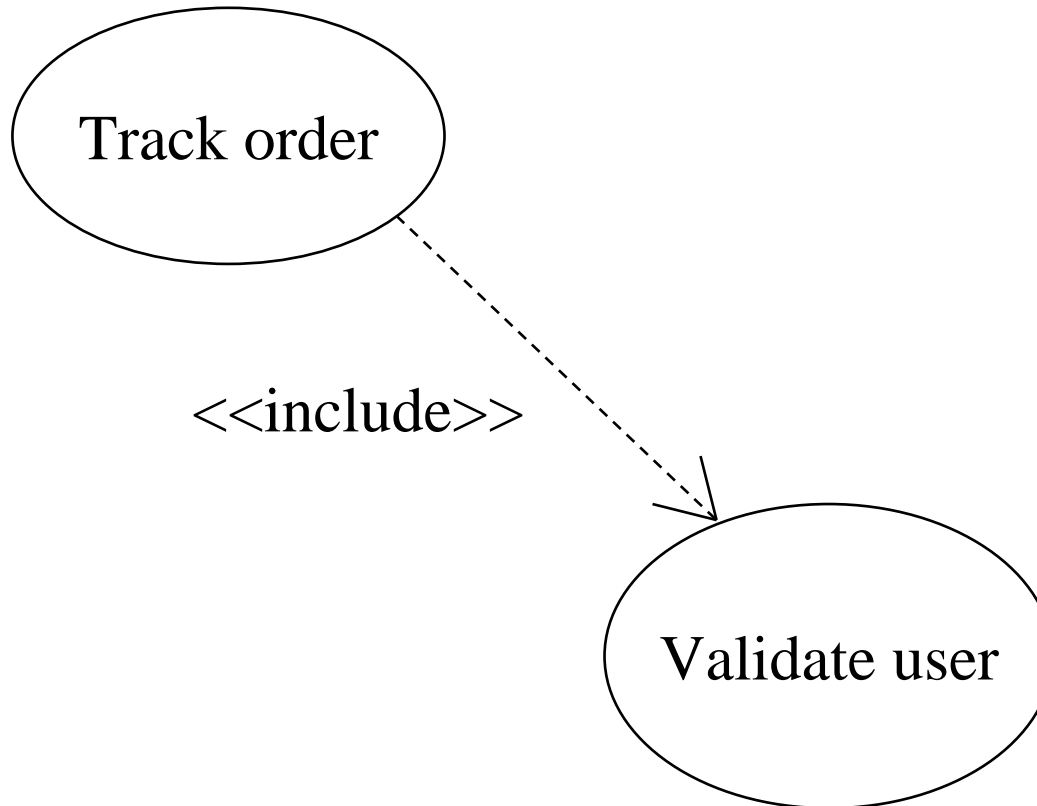
- A use case is always initiated by a primary actor.
- The actor must directly or indirectly be responsible for the initiation of the use case.
- A use case must deliver a tangible result back to the user.
 - Recall: This is defined in the User Guide as:
“an observable result of value to the actor.”

- A use case must be complete.
 - It must be a description of a complete transaction.
 - A common mistake is to partition use cases into smaller use cases that implement each other.
- A use case doesn't actually complete until it produces an end result.
 - This may involve complex dialogue with the actor.

<<include>> (1)

- An <<include>> relationship means that the base use case explicitly uses the behaviour of a use case at a specified location in the base.
- The included use case never stands alone, but is only instantiated as part of some larger base that includes it.
- An <<include>> relationship is used to avoid rewriting the same behaviour several times, i.e. common behaviour is placed in<<include>> use cases.

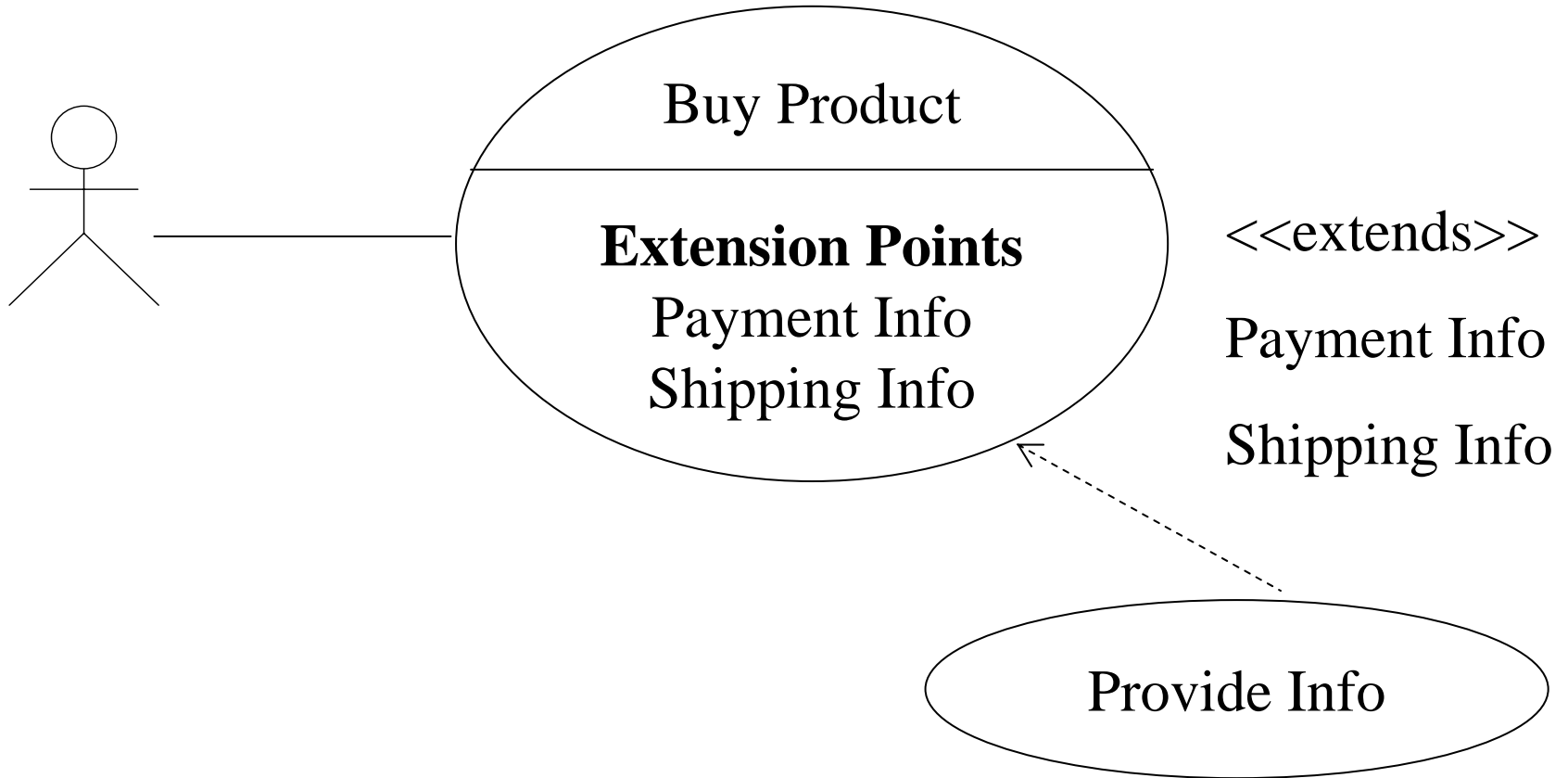
<<include>> (2)



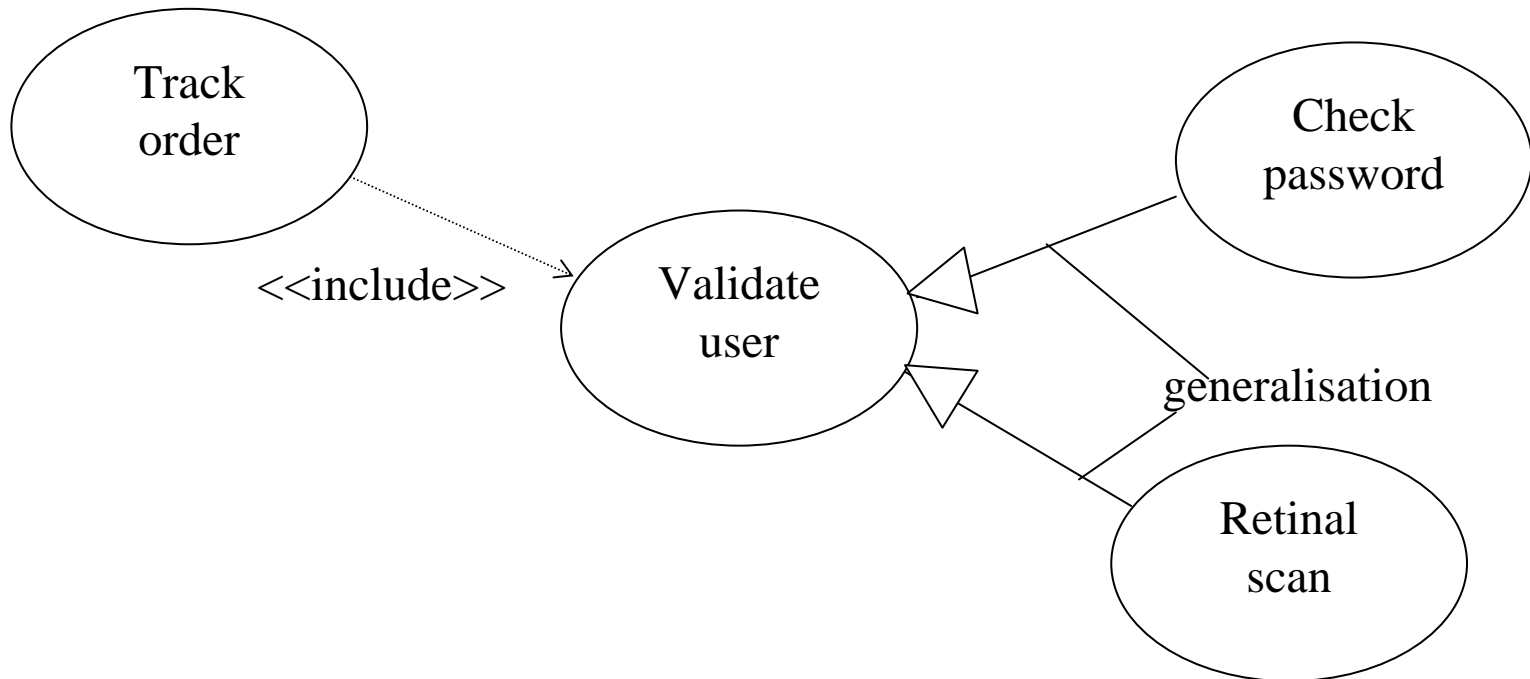
<<extends>> (1)

- The <<extend>> association represents an exception or alternative course of action in a given use case.
- It can only happen dependent on a certain condition or state of the base use case.
- The extension use case must be completed before control is returned to the base use case.

<<extends>> (2)



- Use case generalisation is the same (idea) as class generalisation.
- This is an inheritance structure that is the same, in essence, as the `<<include>>` relationship.
- The child use case inherits all the functionality of the parent use case and can override the parent use case.



- Use cases designed to be a basis of communication between customer and developer to work out the functional requirements of the system.
- An example of a scenario based approach.
 - As a consequence make sure that the diagrams you have produced meet their needs.
- ‘Walk’ the customer through the use case
 - and if possible ‘walk’ those who will play a specific role in the system through it as well.

- The walkthrough idea can be taken one step further with certain forms of validation
 - this is particularly true of the description.
- For example, **enactable** models of use cases can be produced so that the user is forced to step through each action or event (and such that the implications of the events are made clearer).
 - This is an idea taken from process modelling
 - Will examine both enactable process and use case models.

Use your use case

- A use case is actually realised in later design stages as one or more classes, often known as a *collaboration*.
 - These may be given life as *sequence*, *activity*, *collaboration* or *state* diagrams.
- This link can be traceably documented through the use of an <<implements>> relationship.

- Avoid *analysis paralysis*.
- Use cases are really orthogonal to OO. (*Why?*)
- The process by which use cases are selected, and by which scenarios are developed from them, is only vaguely defined.
- Systematic guidance of what specific scenarios to elaborate within a given use case is completely missing.
- *UML have gone mad on use case relationships - don't religiously use them all.*

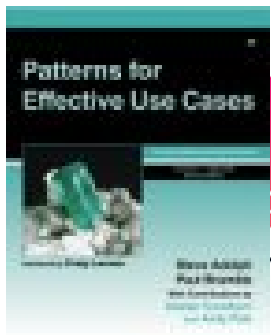
- UML has no systematic procedure to validate the use cases against the requirements,
 - nor to feed the results back in order to expand the scenarios,
 - nor to refine or correct the requirements.
- *Though some promising work on guidelines and dependencies.*
- Use cases and scenarios are hardly supported by present UML-oriented tools.
- *Though again we will examine some (including home grown) approaches.*

Write the Use Case

- Don't waste a lot of time with the diagram notation and what type of arrows to use.
 - Be consistent with the style you pick.
- **WRITE THE USE CASE - THIS IS THE KEY TO SUCCESS.**
 - Writing the use case is where we describe the behaviour within a use case ellipse.
 - These are use case descriptions.

- Typically (though may not be) a form of structured text.
- As we shall see, there is much debate about guidelines for both their content and structure.
- Most agree the general form akin to:
 - Use Case name:
 - Actors:
 - Context:
 - Main flow of events
 - Alternative flows of events

- Use cases allow understanding and common language among stakeholders.
- Use cases are **VERY POPULAR** so you **WILL** come across them.
- Use cases allow one to describe interactions at the interface of the machine.
- However, the diagram is more a partitioning.
- The description (really) acts as a specification.

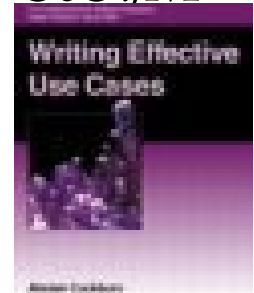


Patterns for Effective Use Cases: Steve Adolph,
Paul Bramble, Alistair Cockburn, Andy Pols



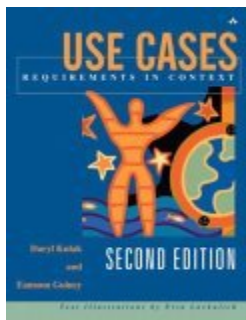
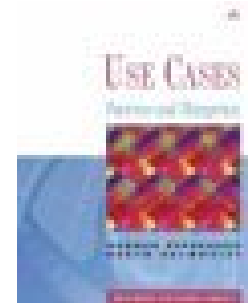
SoSyM

Writing Effective Use Cases: Alistair Cockburn



Scenarios Stories and Use Cases: Through the
Systems Development Life-Cycle: I. Alexander,
N. Maiden

Use Cases: Patterns and Blueprints: Gunnar
Övergaard, Karin Palmkvist



Use Cases, Requirements in Context: Daryl
Kulak, Eamonn Guiney