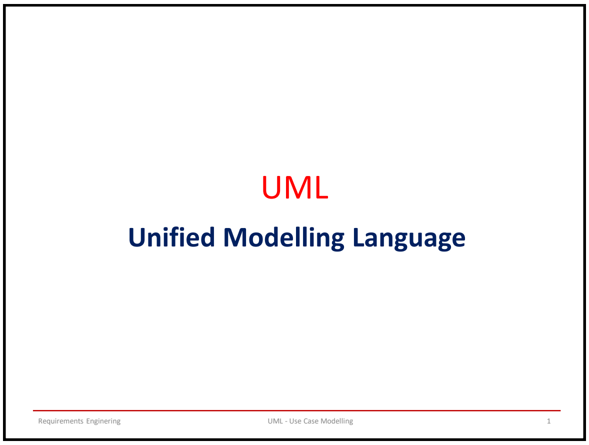
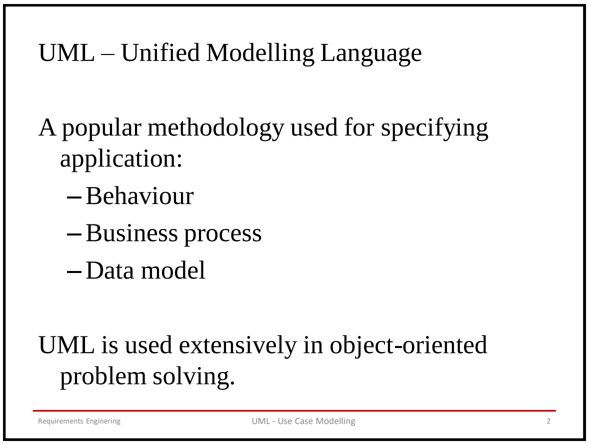


UML

Unified Modelling Language

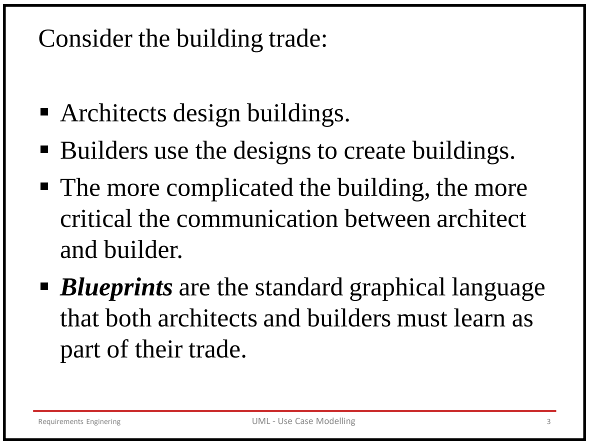


UML – Unified Modelling Language

A popular methodology used for specifying application:

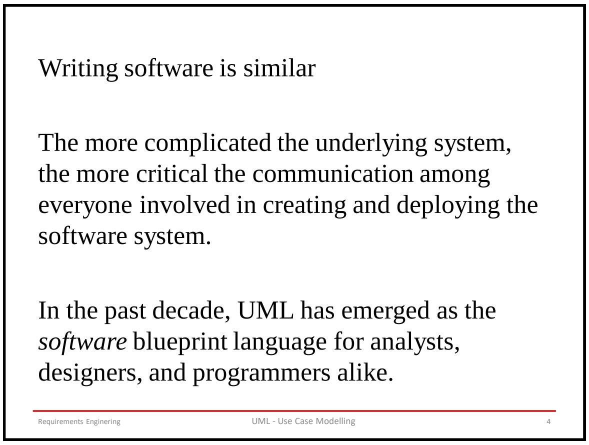
- Behaviour
- Business process
- Data model

UML is used extensively in object-oriented problem solving.



Consider the building trade:

- Architects design buildings.
- Builders use the designs to create buildings.
- The more complicated the building, the more critical the communication between architect and builder.
- ***Blueprints*** are the standard graphical language that both architects and builders must learn as part of their trade.



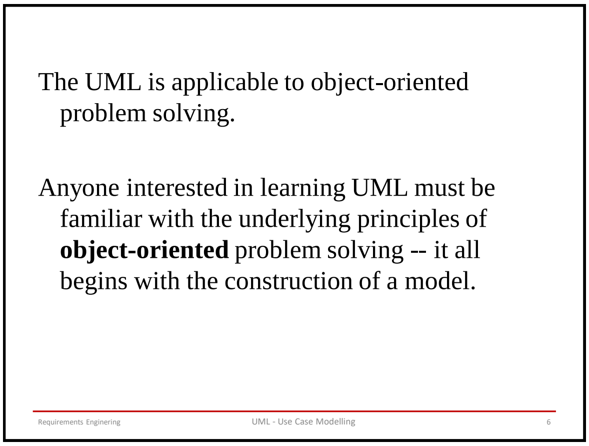
Writing software is similar

The more complicated the underlying system, the more critical the communication among everyone involved in creating and deploying the software system.

In the past decade, UML has emerged as the *software* blueprint language for analysts, designers, and programmers alike.

UML is now an accepted component of the software engineering industry.

UML gives everyone from the business analyst, to the designer, to the programmer a common vocabulary to talk about software design.



The UML is applicable to object-oriented problem solving.

Anyone interested in learning UML must be familiar with the underlying principles of **object-oriented** problem solving -- it all begins with the construction of a model.

There are many UML Modelling Diagrams

- Use case diagrams
- Class diagrams
- Object diagrams
- Sequence diagrams
- Collaboration diagrams
- State chart diagrams
- Activity diagrams
- Component diagrams
- Deployment diagrams

For Requirements Specification….

- Use case diagrams
- Class diagrams
- Object diagrams
- Sequence diagrams
- Collaboration diagrams
- State chart diagrams
- Activity diagrams
- Component diagrams
- Deployment diagrams

Use Case Diagrams

Describe what a system does from a ***users*** viewpoint. Emphasis is on ***what*** a system does and not ***how***.

Use case diagrams are closely connected to ***scenarios***.

A **scenario** is an example of what happens when someone interacts with the system (software).

Here is a scenario for a doctor’s appointment system:

“When a patient calls the receptionist to make an appointment, the receptionist finds the nearest empty time slot in the appointment book for a requested date and schedules the appointment for that time slot."

This is the ***expected scenario***.

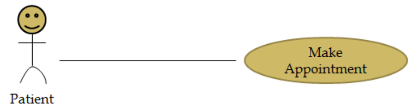
If there are no appointments available on the required date, then an ***alternate scenario*** must be handled by the system.

A ***use case*** is a summary of scenarios for a single task or goal or function.

An ***actor*** is ***who*** or ***what*** initiates the events involved in that task.

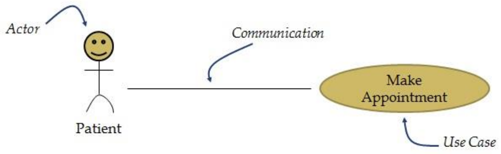
Actors are simply roles that people or objects play.

Use case: ***Make Appointment***

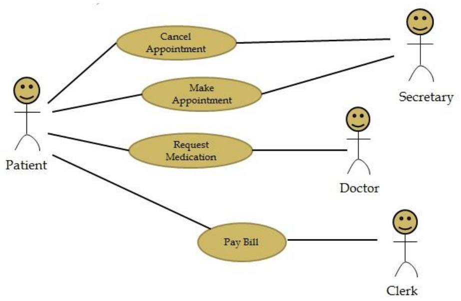


Actors are stick figures.
Use cases are ovals.
Communications are lines that link actors to use cases.

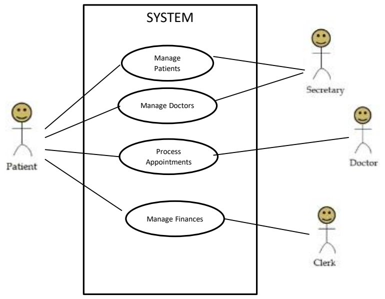
Use case: ***Make Appointment***



A use case diagram is a collection of actors, use cases, and their communications.



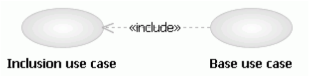
A system level use case is ***high level*** and shows all actors and the ***system boundary***.



Sometimes, one use case includes the functionality of another use case.

In this case, an ***include*** relationship is shown on the model.

The ***base*** use case includes the functionality of another ***inclusion*** use case.



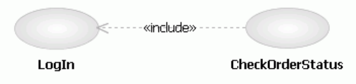
The include relationship supports the ***reuse*** of functionality in a use case model.

The functionality of an inclusion use case is ***always*** performed by the base use case.

Example:

An e-commerce system provides customers with the option of checking the status of their orders.

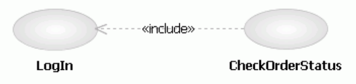
This behaviour is modelled with a base use case called *CheckOrderStatus* that has an inclusion use case called *LogIn*.



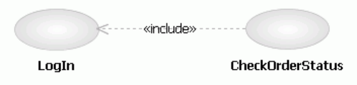
The ***LogIn*** use case is shown as a separate inclusion use case because it contains behaviours that several other use cases (functions) in the system use:

View bill

Update profile

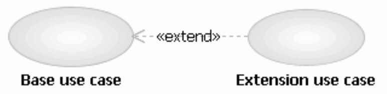


An include relationship points from the *CheckOrderStatus* use case to the *LogIn* use case to indicate that the CheckOrderStatus use case ***always*** includes the behaviours in the LogIn use case.



An ***extend*** relationship is used to specify that one use case (extension) extends the behaviour of another use case (base).

This type of relationship reveals details about a system or application that are typically hidden in a use case.



The extend relationship specifies that the incorporation of the extension use case is dependent on what happens when the base use case executes.

The extension use case ***might*** or ***might not*** be used

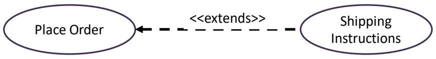
The extension use case owns the extend relationship. You can specify several extend relationships for a single base use case.

The extension use case can access and modify the attributes of the base use case;

However, the base use case is not aware of the extension use case and, therefore, cannot access or modify the attributes and operations of the extension use case.

Example:

An e-commerce system in which you have a base use case called ***Place Online Order*** that has an extending use case called ***Specify Shipping Instructions***.



An extend relationship points from the *Specify Shipping Instructions* use case to the *Place Order* use case to indicate that the behaviours in the *Specify Shipping Instructions* use case are ***optional*** and only occur in certain circumstances.

Use case diagrams have three main functions:

- Determining requirements
- Communicating with clients
- Generating test cases

Requirements Use-Case Modelling

Objective is to elicit and analyse sufficient requirements information to prepare a model of ***what*** is required from the user but does not include details about ***how*** the system is to be built and implemented

Not all facts will be identified initially, but will be identified by using iterative, incremental development.

Use-Case modelling requires the following steps:

1. Identify business actors
2. Identify business requirements use cases
3. Construct use-case model diagram(s)
4. Document business requirements in ***use-case narratives***

1. Identify Business Actors

Why identify actors first?

- Focus on the actors to understand how the system will be used (not built)
- Helps to refine and further define the scope (functions) and boundaries of the system
- Actors determine the completeness of the system requirements

- Identifies candidates which can be later interviewed and observed to complete use-case modelling.
- These candidates can be used to validate and verify completed use cases.

Sources for identifying actors might include:

- Context diagram showing system scope and boundaries
- Existing documentation and user manuals
- Minutes of project meetings and workshops
- Existing requirements documents

Questions to be asked:

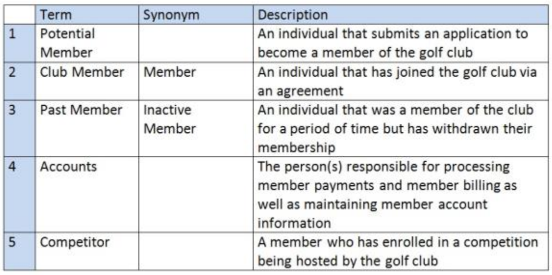
- Who or what provides inputs to the system?
- Who or what receives outputs from the system?
- Are interfaces required to other systems?
- Are there any events that are automatically triggered at a predetermined time?
- Who will maintain information in the system?

When actors are identified, create a textual definition of the actor from the users perspective.

Document the actors in an ***Actor Glossary.***

An actor glossary **must** be included in your design document.

Actor glossary for a golf club administration system



	Term	Synonym	Description
1	Potential Member		An individual that submits an application to become a member of the golf club
2	Club Member	Member	An individual that has joined the golf club via an agreement
3	Past Member	Inactive Member	An individual that was a member of the club for a period of time but has withdrawn their membership
4	Accounts		The person(s) responsible for processing member payments and member billing as well as maintaining member account information
5	Competitor		A member who has enrolled in a competition being hosted by the golf club

2. Identify Business Requirements Use Cases

- A typical information system will have many use cases (possibly hundreds!)
- Requirements analysis should aim to document the most critical, complex and important use cases (essential use cases)
- A business requirements use case captures the interactions between the software and a user which is free of technological and implementation detail

- Examining actors and how they will use the system is a good technique for identifying business requirements use cases.

Questions to ask:

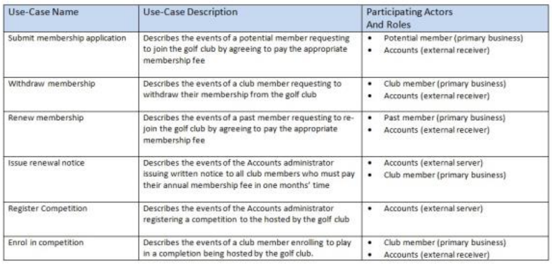
- What are the main tasks of the actor?
- What information does the actor need from the system?
- What information does the actor provide to the system?
- Does the system need to inform the actor of any changes or events that have occurred?
- Does the actor need to inform the system of any changes or events that have occurred?

Business requirements use cases must be documented .

A ***use-case glossary*** is one way that this can be done.

An Use Case glossary **must** be included in your design document.

Use Case glossary for a golf club administration system



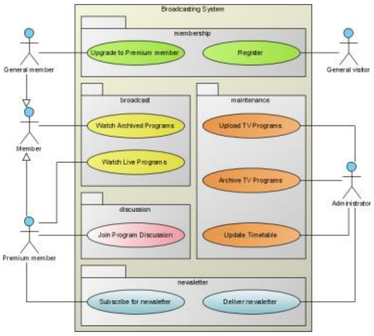
Use-Case Name	Use-Case Description	Participating Actors And Roles
Submit membership application	Describes the events of a potential member requesting to join the golf club by agreeing to pay the appropriate membership fee	• Potential member (primary business) • Accounts (external receiver)
Withdraw membership	Describes the events of a club member requesting to withdraw their membership from the golf club	• Club member (primary business) • Accounts (external receiver)
Renew membership	Describes the events of a past member requesting to re-join the golf club by agreeing to pay the appropriate membership fee	• Past member (primary business) • Accounts (external receiver)
Issue renewal notice	Describes the events of the Accounts administrator issuing written notice to all club members who must pay their annual membership fee in one months' time	• Accounts (external server) • Club member (primary business)
Register Competition	Describes the events of the Accounts administrator registering a competition to the hosted by the golf club	• Accounts (external server)
Enrol in competition	Describes the events of a club member enrolling to play in a completion being hosted by the golf club.	• Club member (primary business) • Accounts (external receiver)

Source: Google

3. Construct Use-Case Model Diagram

- Use case diagram can now be drawn to define system scope and boundaries
- Diagram should include use cases listed in the use case glossary

- Use cases may be grouped into business subsystems (UMLs package symbol):
  - Represent logical functional areas of the business process
  - Clarifies system architecture
  - Helps identify development strategy



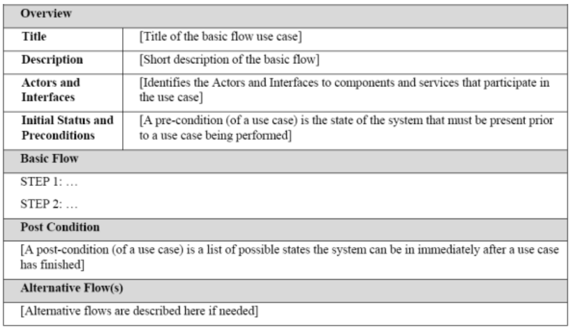
4. Document Business Requirements in Use-Case Narratives

A Use Case narrative is a **textual description** of a business requirement.

Most organisations have a defined **template** which is completed for each business requirement.

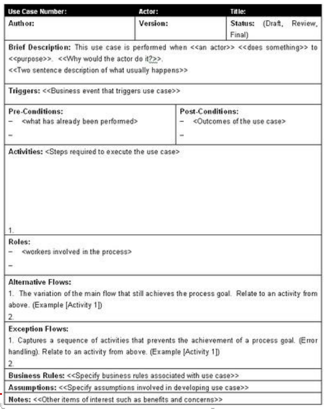
Production involves:

- Initial draft at high level
- Return to each use case and expand to a fully documented business requirement



Overview	
Title	[Title of the basic flow use case]
Description	[Short description of the basic flow]
Actors and Interfaces	[Identifies the Actors and Interfaces to components and services that participate in the use case]
Initial Status and Preconditions	[A pre-condition (of a use case) is the state of the system that must be present prior to a use case being performed]
Basic Flow	
STEP 1: … STEP 2: …	
Post Condition	
[A post-condition (of a use case) is a list of possible states the system can be in immediately after a use case has finished]	
Alternative Flow(s)	
[Alternative flows are described here if needed]	

Source: Google!



Use Case Number:	**Actor:**	**Title:**
Author:	**Version:**	**Status:** (Draft, Review, Final)
Brief Description: This use case is performed when <<an actor>> <<does something>> to <<purpose>>. <<Why would the actor do it?>>. <<Two sentence description of what usually happens>>		
Triggers: <<Business event that triggers use case>>		
Pre-Conditions: – <what has already been performed> –	**Post-Conditions:** – <Outcomes of the use case> –	
Activities: <Steps required to execute the use case> 1.		
Roles: – <workers involved in the process> –		
Alternative Flows: 1. The variation of the main flow that still achieves the process goal. Relate to an activity from above. (Example [Activity 1]) 2.		
Exception Flows: 1. Captures a sequence of activities that prevents the achievement of a process goal. (Error handling). Relate to an activity from above. (Example [Activity 1]) 2.		
Business Rules: <<Specify business rules associated with use case>>		
Assumptions: <<Specify assumptions involved in developing use case>>		
Notes: <<Other items of interest such as benefits and concerns>>		