



CO1605

Systems Analysis and Database Design

Julie Allen

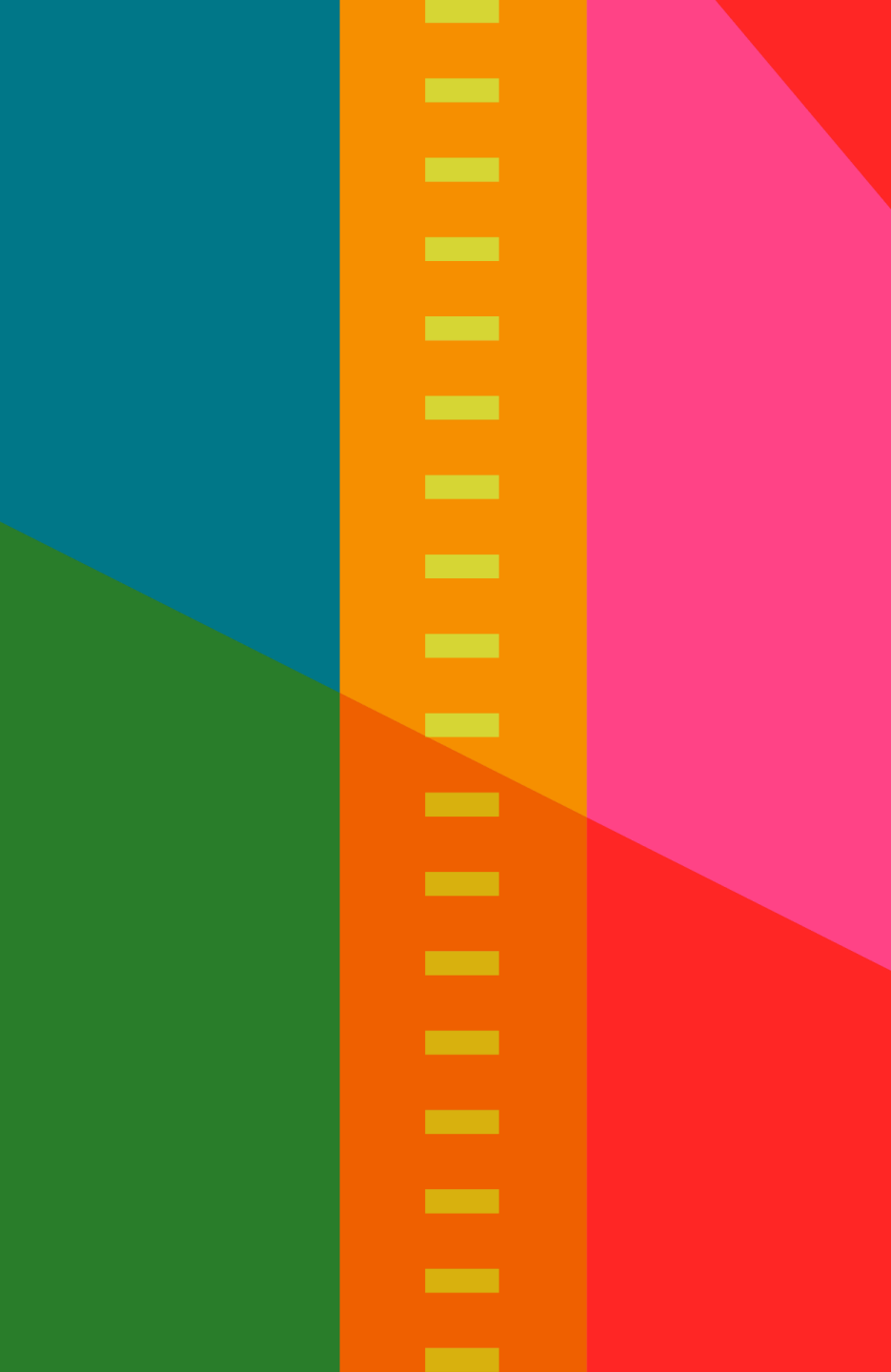
JAllen17@uclan.ac.uk

C&T Building Room CM214

UML: Unified Modelling Language



Unified
Modeling
Language



Learning Outcomes

Students will be able to:

- Understand the concepts of UML
- Identify elements of a Use Case Diagram
- Understand the terminology associated with Use Case Diagrams
- Draw a Use Case Diagram

What is UML?

- Stands for: **Unified Modelling Language**
- A standardised modelling language
- A set of **13 diagrams** to aid in the analysis, design and implementation of software systems
- Helps us to visualise the structure, behaviour and interactions of a software system

The Benefits of UML

To visualise
the system as
it is or as we
want it to be

To give a
template to
guide for
construction
of a system

To specify the
structure or
behaviour of a
system

To document
decisions

A picture paints a thousand words

13 Different Types of Diagram...

Structure

- Class
- Component
- Deployment
- Composite Structure
- Object
- Package

Behaviour

- Use Case
- Activity
- State Transition
- Sequence
- Timing
- Communication
- Interaction Overview

We will be creating...

Use Case Diagram

- **Behaviour diagram**
- Models the functions & environment
- Which users interact with the system

Activity Diagram

- **Behaviour diagram**
- Shows the flow of a process
- Behaviour can be procedural and parallel

Class Diagram

- **Structure diagram**
- Models the structure of the system

A good diagram...



Has just **the right amount** of detail and structure



Represents what is important for the current task



Is rich in meaning



Models functional and non-functional requirements



Is accurate, fit for purpose and unambiguous



UML Software

- **Microsoft Visio - this is the software we will use**
- ArgoUML
- Rational Rose Technical Developer
- Visual Paradigm
- SELECT
- Poseidon
- And many more...

Uses of UML

Forward Engineering

- Building something from a high-level model

Reverse Engineering

- Taking apart a system to see how it works

Used as a Sketch

- To communicate with users/customers

Used as a Blueprint

- Detailed design for programmer to code

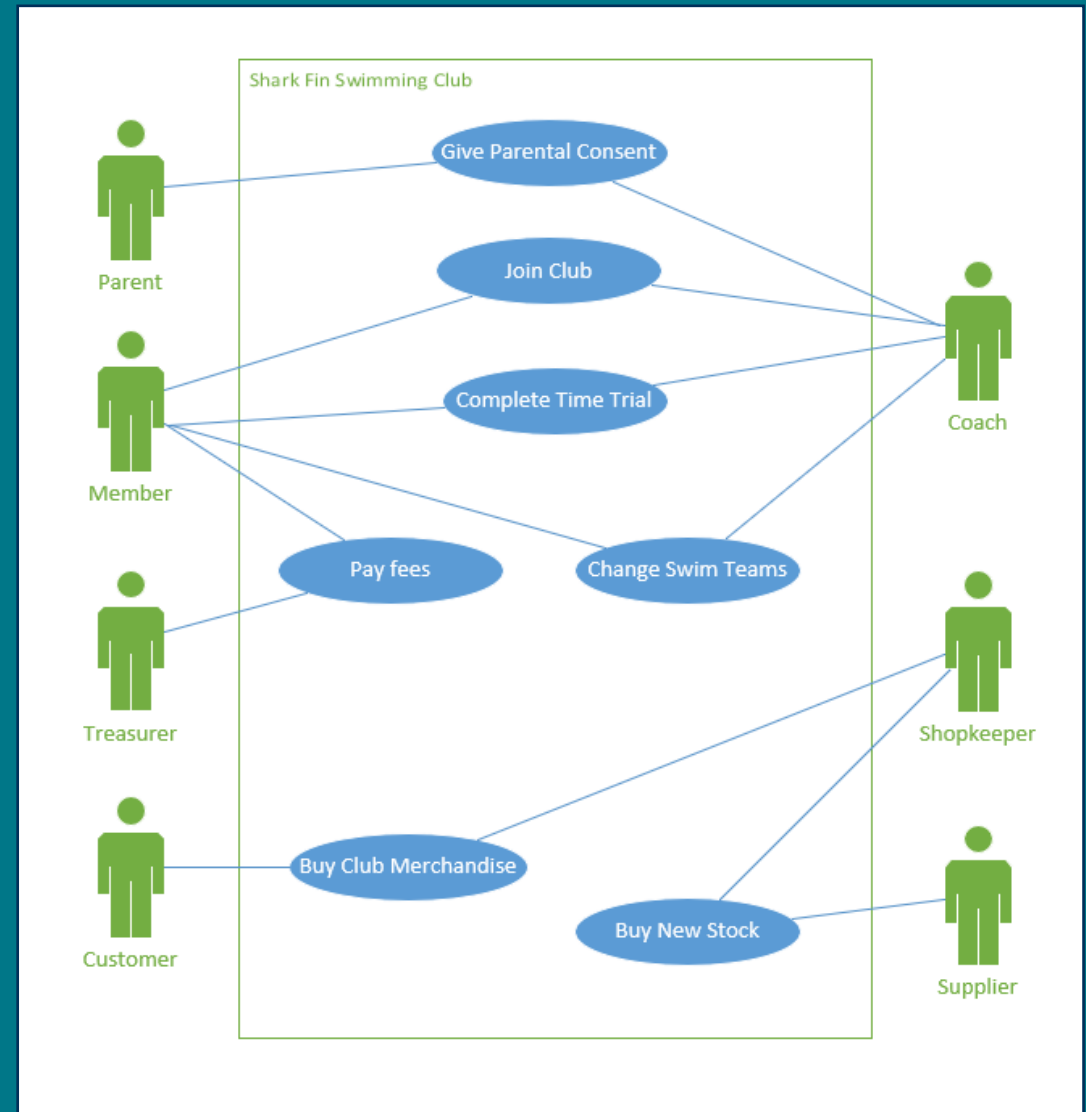
Use Case Diagrams



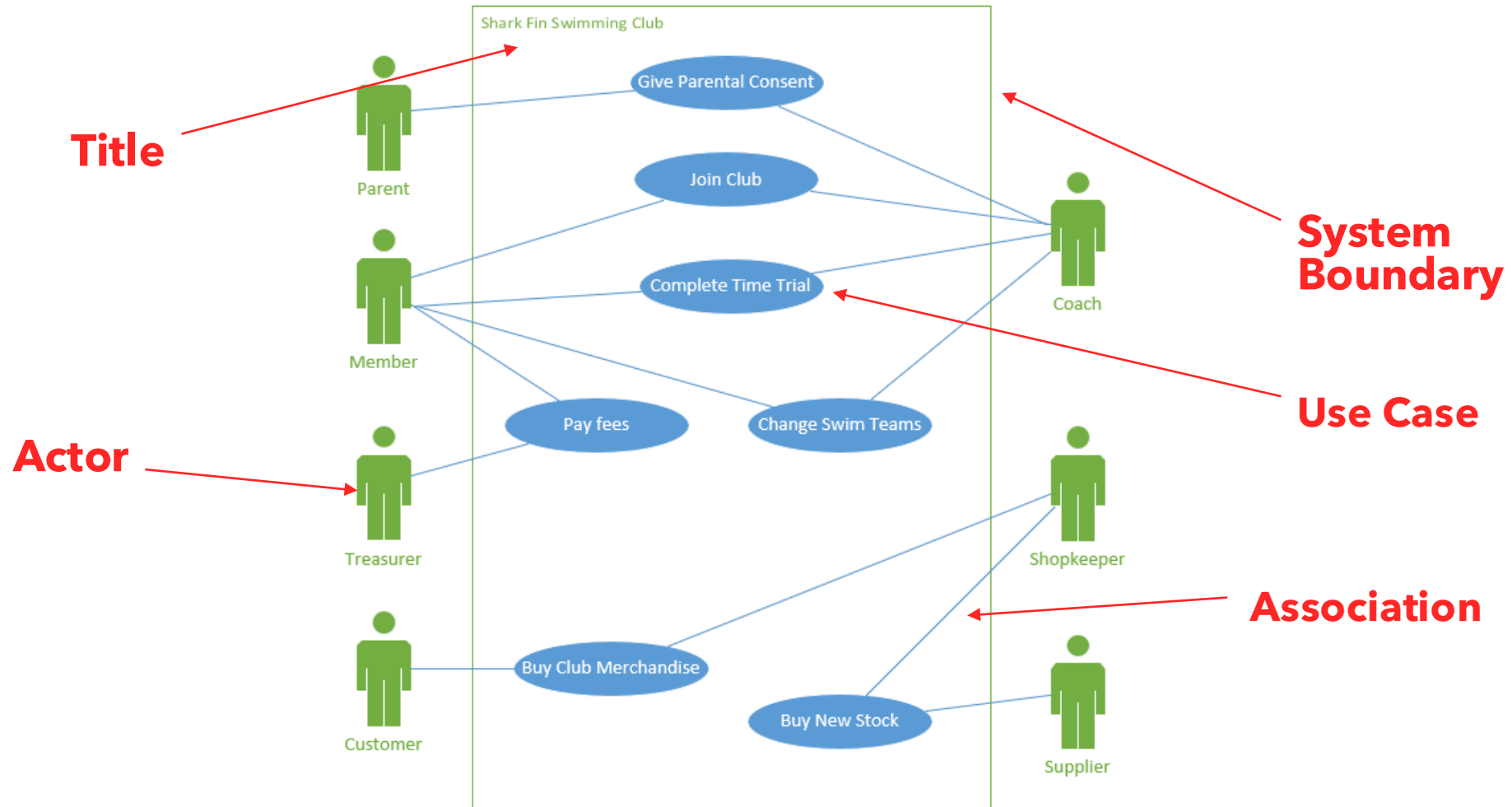
Use Case Diagrams

- Show the different ways in which users might interact with a system
- Capture the functional requirements of a system
- Give a high-level overview - not a lot of detail
- Can document a current system or a new system

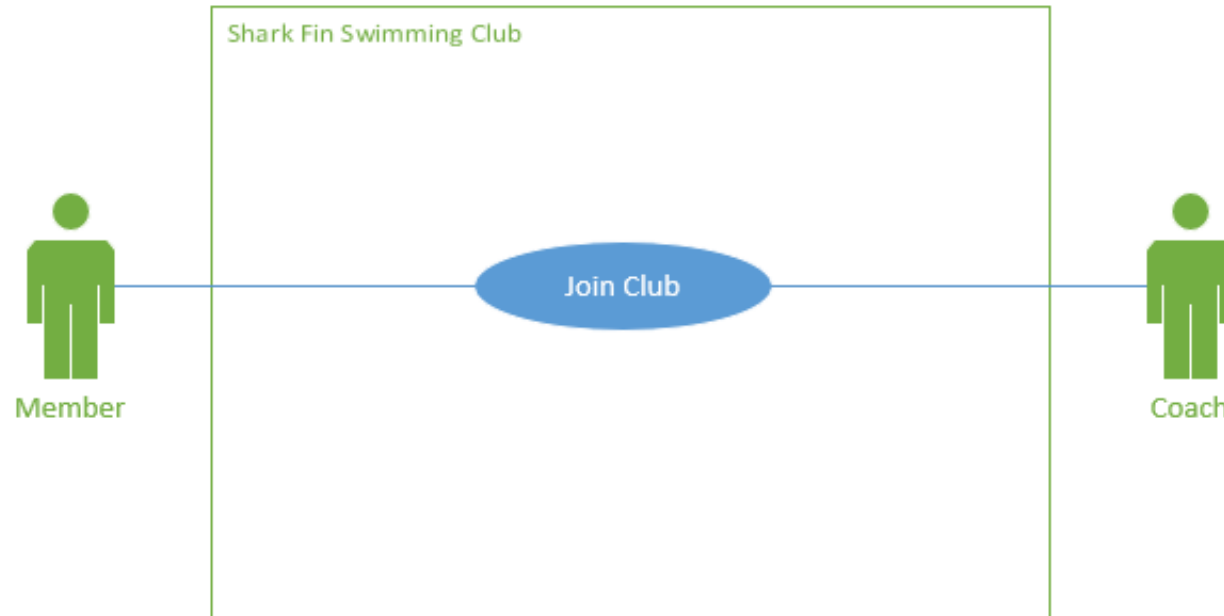
NOTE: The system being modelled does not have to be computerised



Use Case Diagram Notation



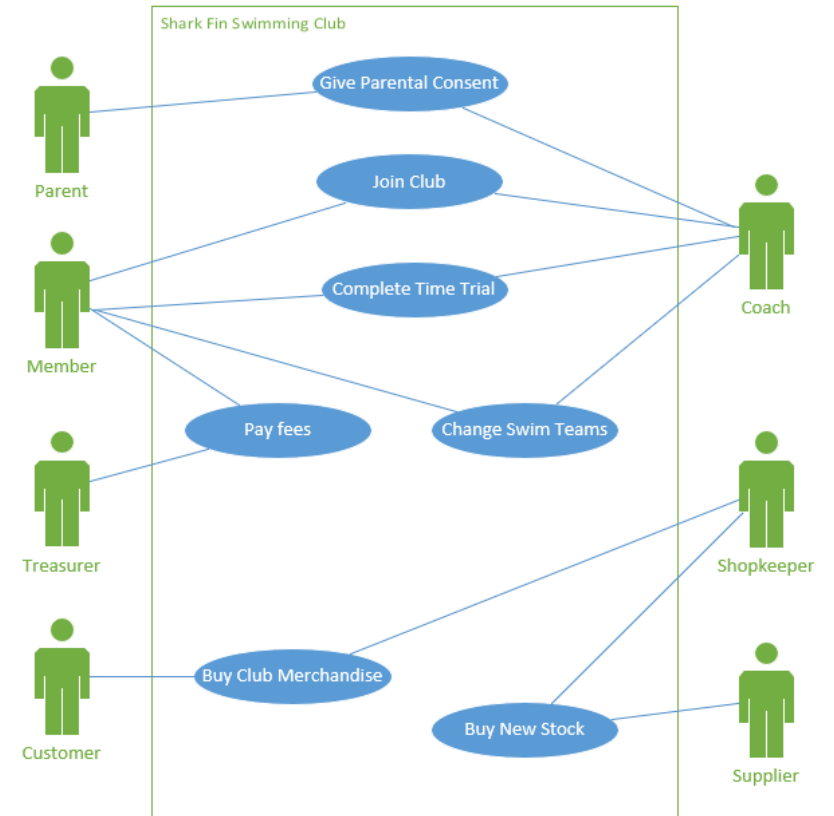
More than one actor can interact with a use case...



The 'Join Club' use case is initiated by a Member, but the Coach also interacts with this use case.

Use Cases

- Use cases are **initiated by an actor**
- They are an end goal of the actor
- They are **event-driven** – the system will perform certain actions in response to the user actions/input
- Each use case contains all of the things that must happen to reach that end goal – the complete functionality
- Shows **what** happens – not **how**



Naming Use Cases

- Described by a **unique name** – usually phrased as an **active verb**
- Should be **unambiguous**
- Should be descriptive, but worded **concisely**



Use Cases - good or bad?



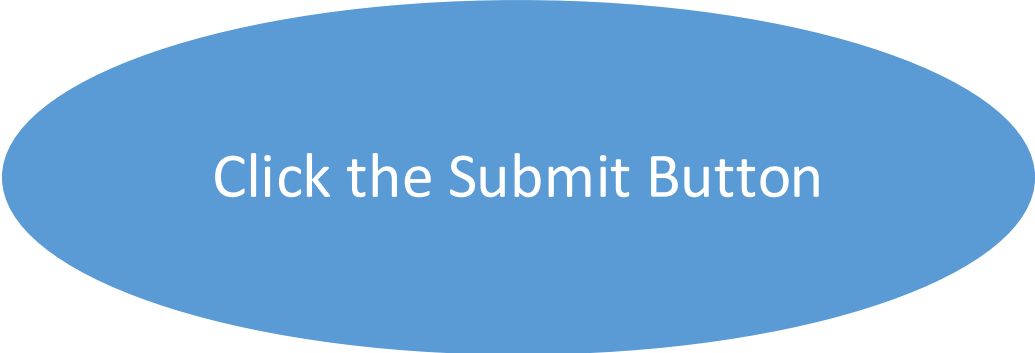
This is ambiguous, join what? ... 'Join Club' is better.

Use Cases – good or bad?

Join the club by phoning and speaking
to the manager of the club

Too wordy - keep it concise.

Use Cases - good or bad?



Click the Submit Button

**Too granular - needs to be an end goal of the user,
not just a step within an end goal.**

Use Cases - good or bad?



**Do what with the stock records? Remember Use Cases model behaviour - some action is performed.
'Update Stock Records' is better.**

Actors

- An actor is not a specific user, but a **role** that a user plays whilst interacting with the system
- An actor uses the system **to achieve a goal**
- They are not just humans; they can be other computers – pieces of hardware or other software systems, or other organisations
- They are named after a **role** that a user (or external system) performs

Each actor can perform many use cases

Each use case can have many actors performing it



Actor

Actors – good or bad?

This actor is named after a person

Actors should be named after roles that are performed



Actors – good or bad?

Better – but manager of what?



Manager

Actors – good or bad?

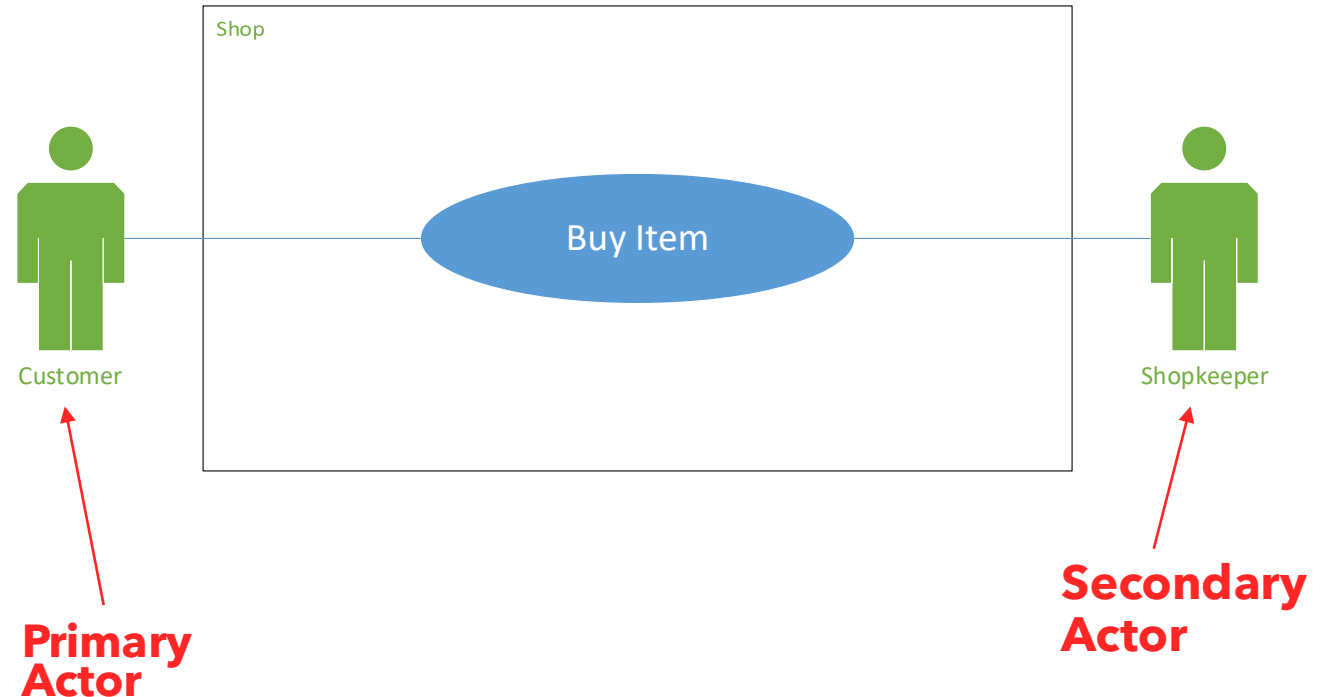
Much Better!



Stock Manager

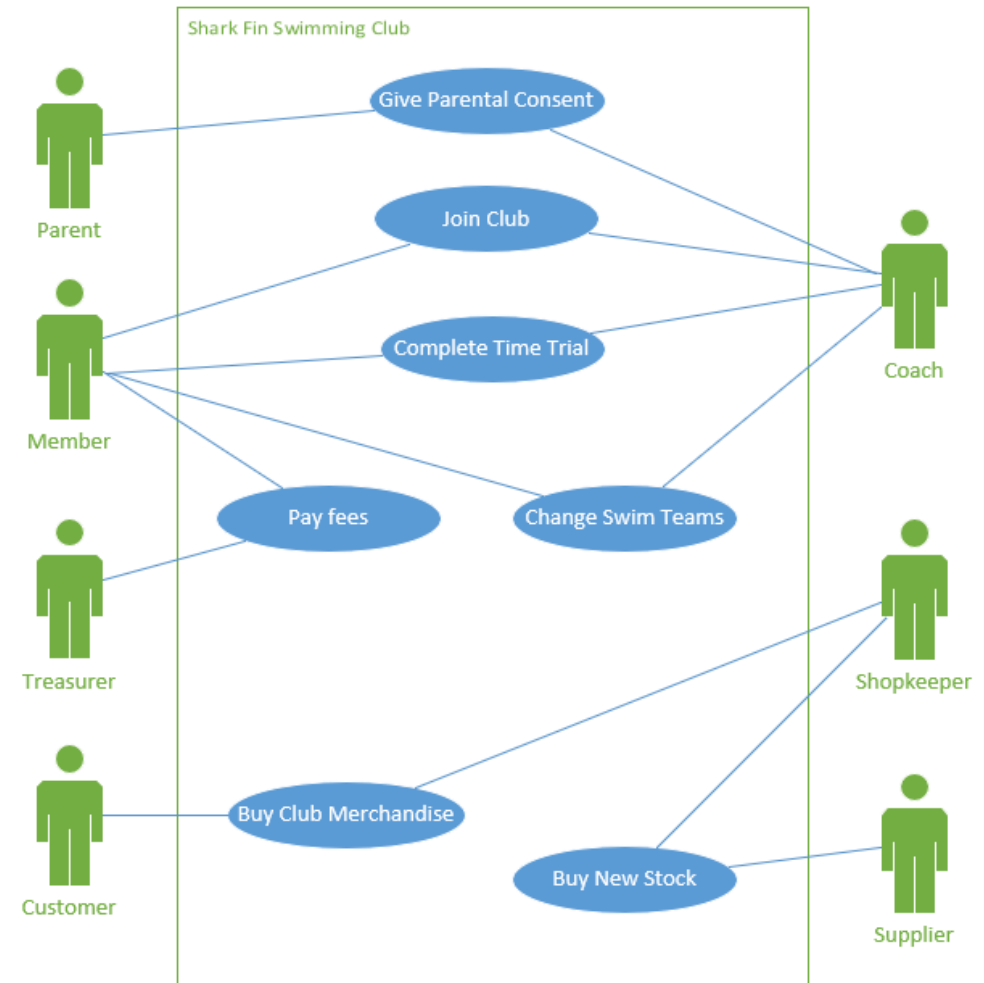
Primary and Secondary Actors

- **Primary Actors** initiate a Use Case
- **Secondary Actors** are involved in a use case but don't initiate it.
- **Primary Actors** are usually placed on the **left side** of the diagram
- **Secondary Actors** are usually placed on the **right side** of the diagram



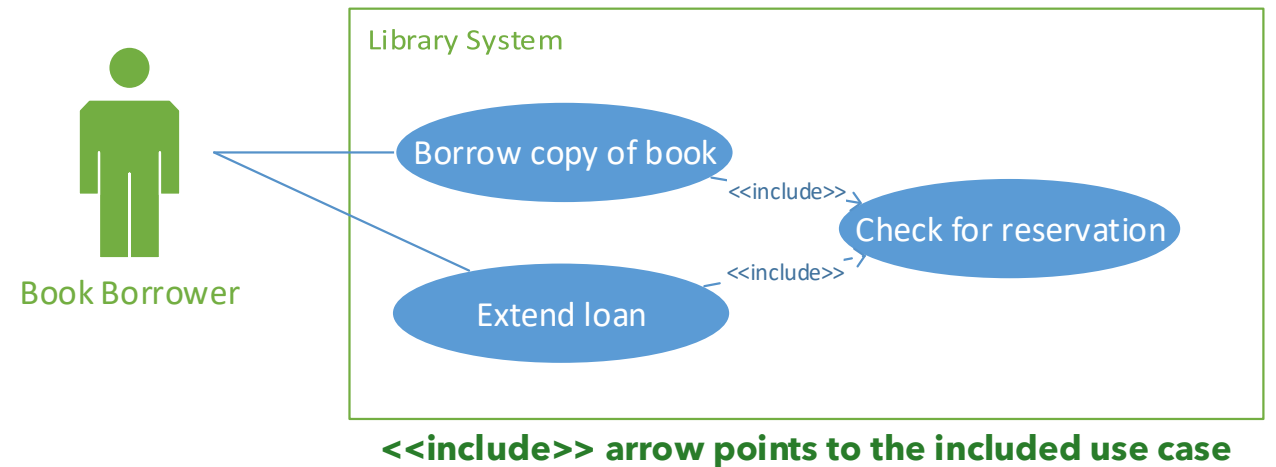
System Boundary

- Shows the **scope** of the system
- Shows **all use cases** included in the system
- **Use cases** are placed **inside** the system boundary
- **Actors** are placed on the **outside**
- When designing systems specifying a boundary can be an issue - the **scope needs to be clarified with the client early on**



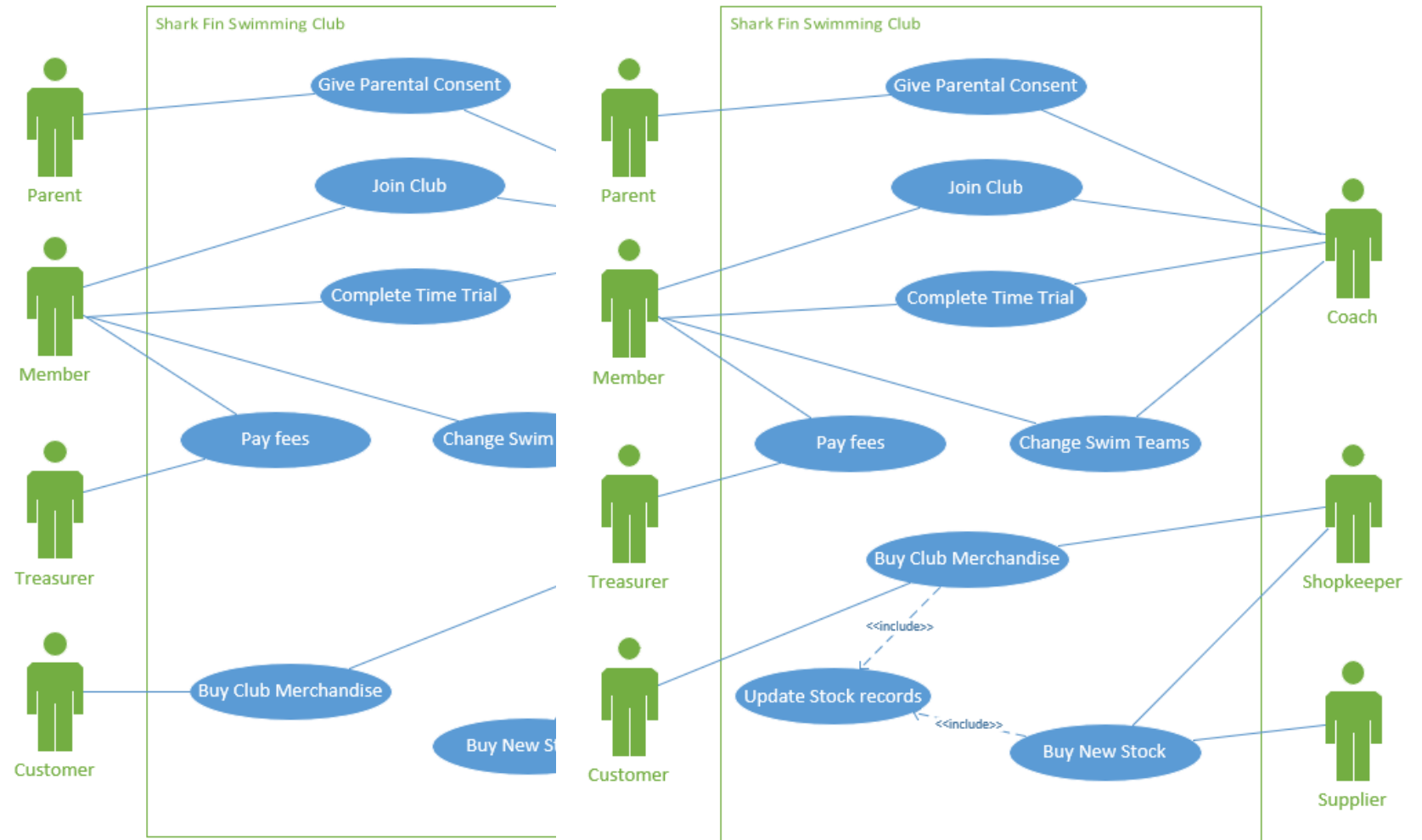
<<INCLUDE>>

- **Common behaviour** can be **separated** and 'called' from multiple use cases
- Shows **common functionality** between 2 or more use cases
- E.g. - In a library system you must check for a reservation:
 - before you borrow a copy of a book**AND**
 - before a loan can be extended



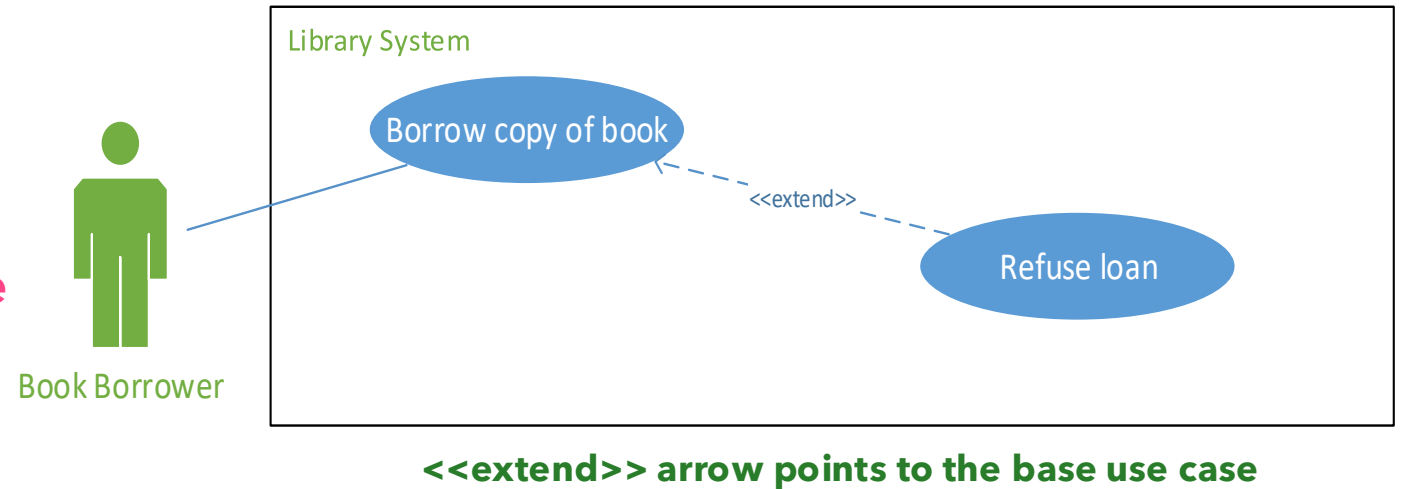
The Base Use Case is incomplete without the included one
The included use case is mandatory

Can we add an <<include>> to the Shark Fin diagram?



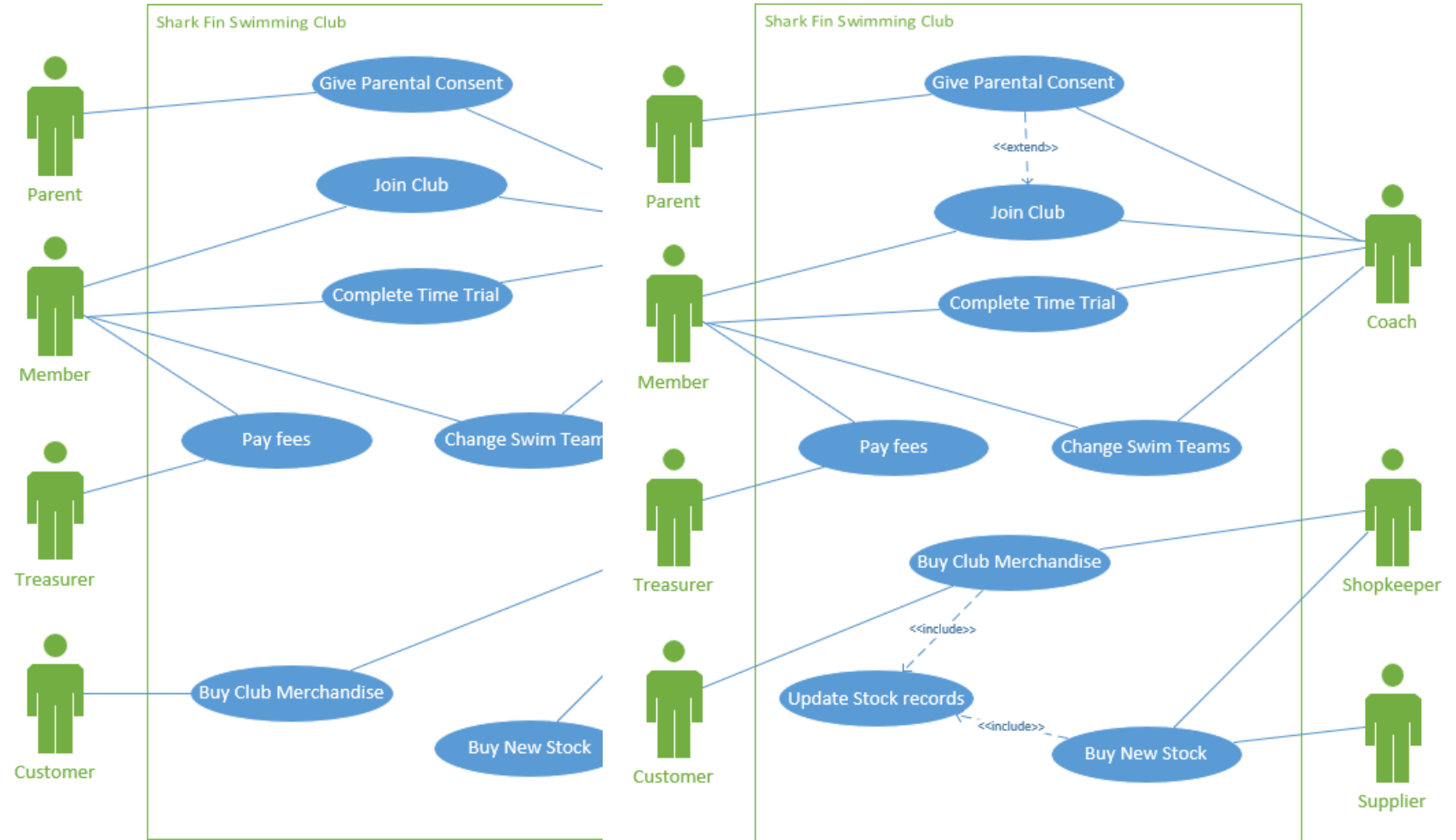
<<EXTEND>>

- **Extending** use case continues behaviour of a **base** use case
- **Most common use...**
- If a use case incorporates two or more significantly different scenarios, e.g., different things happen depending on circumstances, we may decide that it would be clearer to show these as a **base** use case and one or more **subsidiary** cases
- When to use this is a matter of judgement...
 - **Failure Scenarios**
 - **Unusual situations**



**The Base Use Case is complete without the extended one
Extended use case is optional**

Can we add an <<extend>> to the Shark Fin diagram?

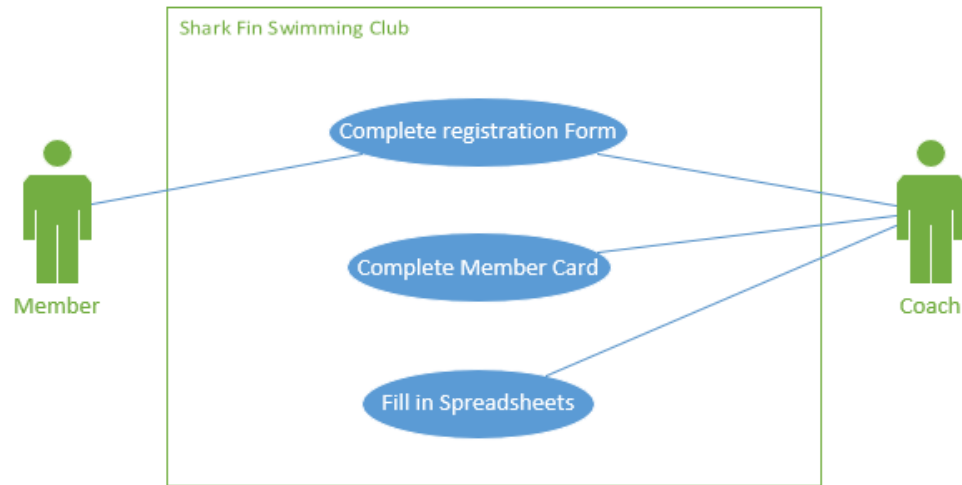


Granularity (level of detail)

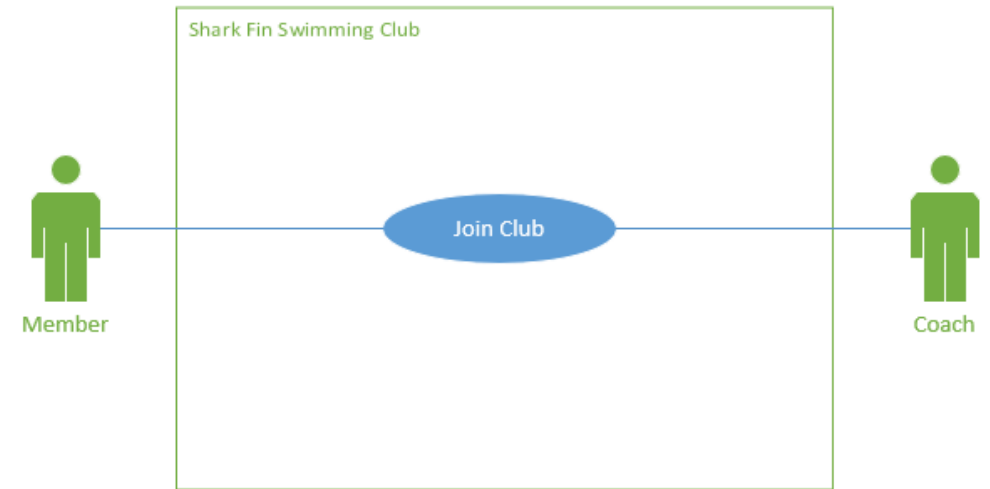
- A diagram should cover everything within the system boundary, but...
- Should give a high-level overview
- Use Cases are end goals of actors
- The detail (functionality) is contained within each use case



Granularity (level of detail)



Too much detail



Appropriate level of detail

DEMO

Any Questions?