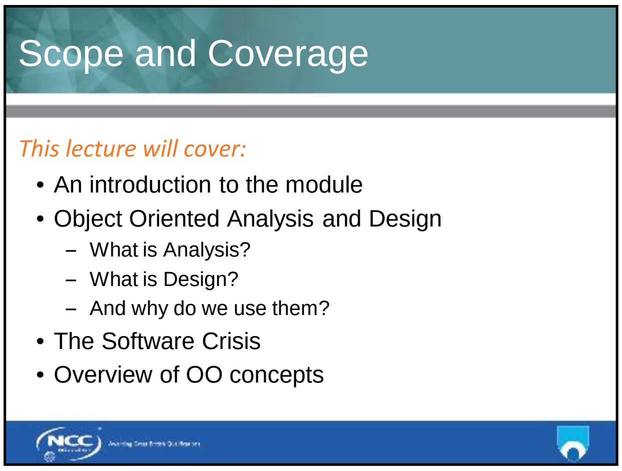
Awarding Great British Qualifications

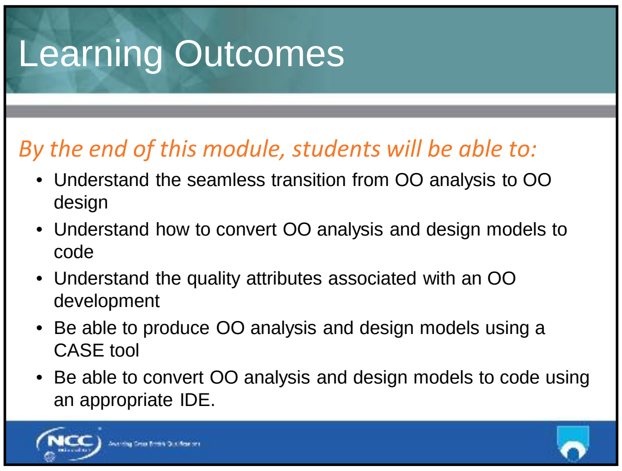
Analysis, Design and Implementation
Topic 1:
Introduction to Module



Scope and Coverage

This lecture will cover:

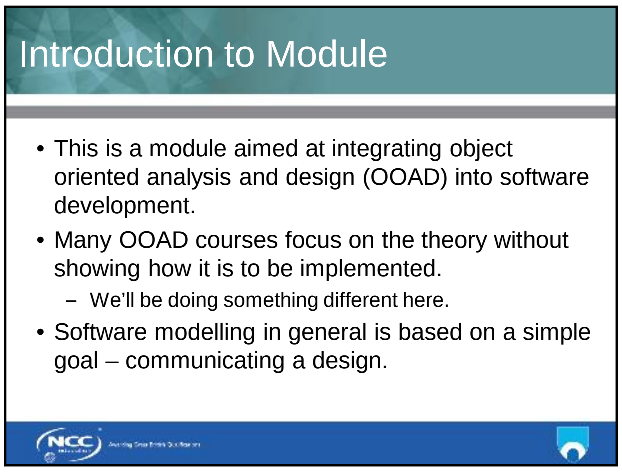
- An introduction to the module
- Object Oriented Analysis and Design
  - What is Analysis?
  - What is Design?
  - And why do we use them?
- The Software Crisis
- Overview of OO concepts



Learning Outcomes

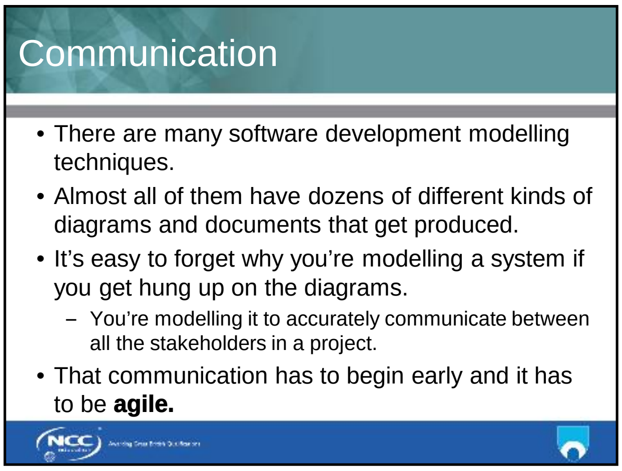
By the end of this module, students will be able to:

- Understand the seamless transition from OO analysis to OO design
- Understand how to convert OO analysis and design models to code
- Understand the quality attributes associated with an OO development
- Be able to produce OO analysis and design models using a CASE tool
- Be able to convert OO analysis and design models to code using an appropriate IDE.



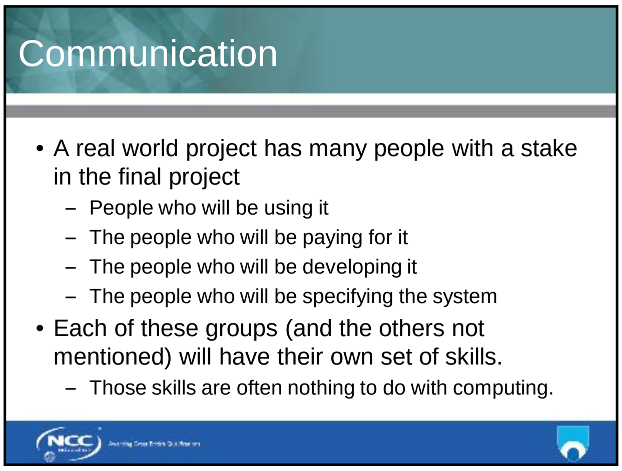
Introduction to Module

- This is a module aimed at integrating object oriented analysis and design (OOAD) into software development.
- Many OOAD courses focus on the theory without showing how it is to be implemented.
  - We'll be doing something different here.
- Software modelling in general is based on a simple goal – communicating a design.



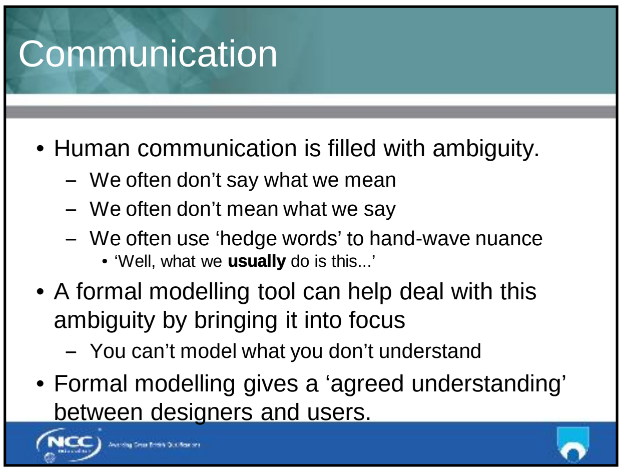
Communication

- There are many software development modelling techniques.
- Almost all of them have dozens of different kinds of diagrams and documents that get produced.
- It's easy to forget why you're modelling a system if you get hung up on the diagrams.
  - You're modelling it to accurately communicate between all the stakeholders in a project.
- That communication has to begin early and it has to be **agile.**



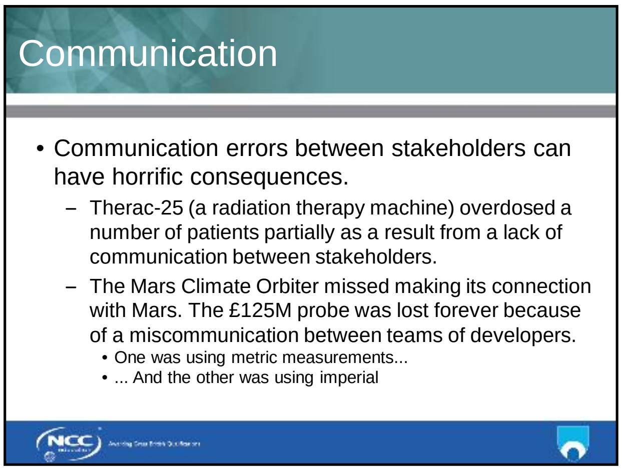
Communication

- A real world project has many people with a stake in the final project
  - People who will be using it
  - The people who will be paying for it
  - The people who will be developing it
  - The people who will be specifying the system
- Each of these groups (and the others not mentioned) will have their own set of skills.
  - Those skills are often nothing to do with computing.



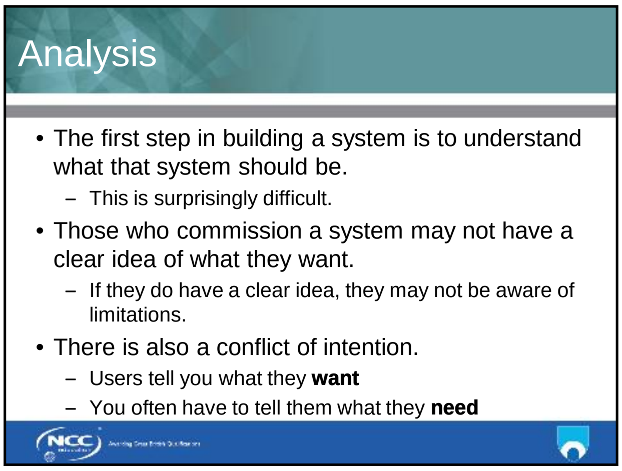
Communication

- Human communication is filled with ambiguity.
  - We often don’t say what we mean
  - We often don’t mean what we say
  - We often use ‘hedge words’ to hand-wave nuance
    - ‘Well, what we **usually** do is this...’
- A formal modelling tool can help deal with this ambiguity by bringing it into focus
  - You can’t model what you don’t understand
- Formal modelling gives a ‘agreed understanding’ between designers and users.



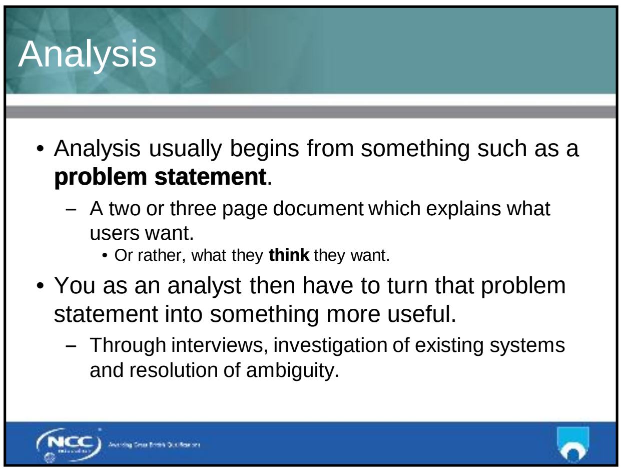
Communication

- Communication errors between stakeholders can have horrific consequences.
  - Therac-25 (a radiation therapy machine) overdosed a number of patients partially as a result from a lack of communication between stakeholders.
  - The Mars Climate Orbiter missed making its connection with Mars. The £125M probe was lost forever because of a miscommunication between teams of developers.
    - One was using metric measurements...
    - ... And the other was using imperial



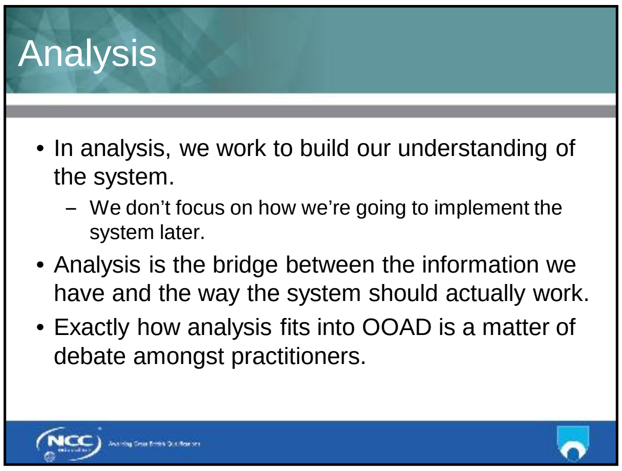
Analysis

- The first step in building a system is to understand what that system should be.
  - This is surprisingly difficult.
- Those who commission a system may not have a clear idea of what they want.
  - If they do have a clear idea, they may not be aware of limitations.
- There is also a conflict of intention.
  - Users tell you what they **want**
  - You often have to tell them what they **need**



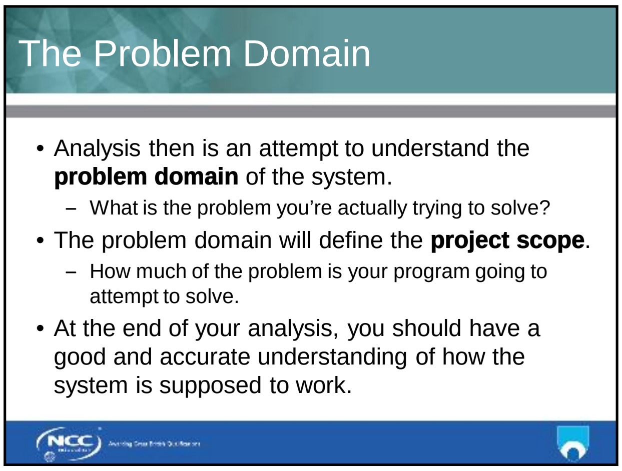
Analysis

- Analysis usually begins from something such as a **problem statement**.
  - A two or three page document which explains what users want.
    - Or rather, what they **think** they want.
- You as an analyst then have to turn that problem statement into something more useful.
  - Through interviews, investigation of existing systems and resolution of ambiguity.



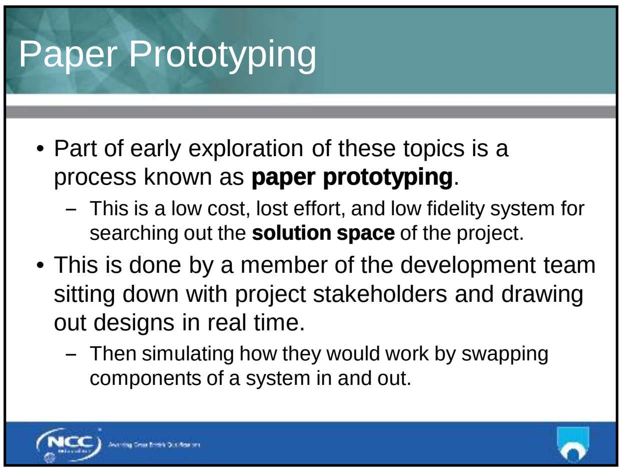
Analysis

- In analysis, we work to build our understanding of the system.
  - We don't focus on how we're going to implement the system later.
- Analysis is the bridge between the information we have and the way the system should actually work.
- Exactly how analysis fits into OOAD is a matter of debate amongst practitioners.



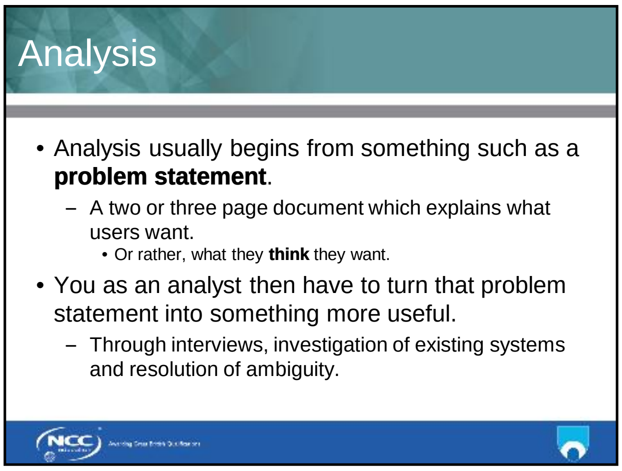
The Problem Domain

- Analysis then is an attempt to understand the **problem domain** of the system.
  - What is the problem you're actually trying to solve?
- The problem domain will define the **project scope**.
  - How much of the problem is your program going to attempt to solve.
- At the end of your analysis, you should have a good and accurate understanding of how the system is supposed to work.



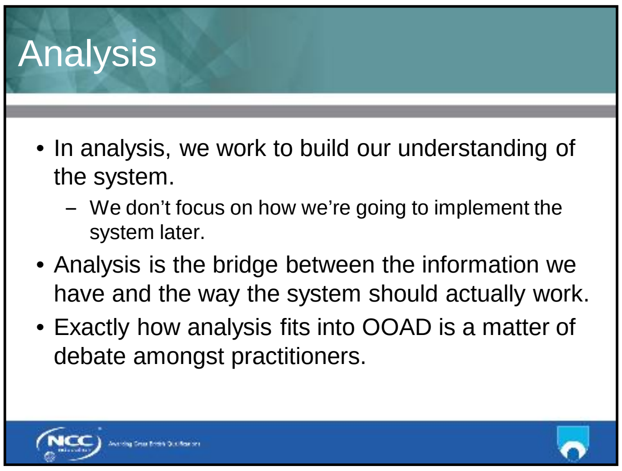
Paper Prototyping

- Part of early exploration of these topics is a process known as **paper prototyping**.
  - This is a low cost, lost effort, and low fidelity system for searching out the **solution space** of the project.
- This is done by a member of the development team sitting down with project stakeholders and drawing out designs in real time.
  - Then simulating how they would work by swapping components of a system in and out.



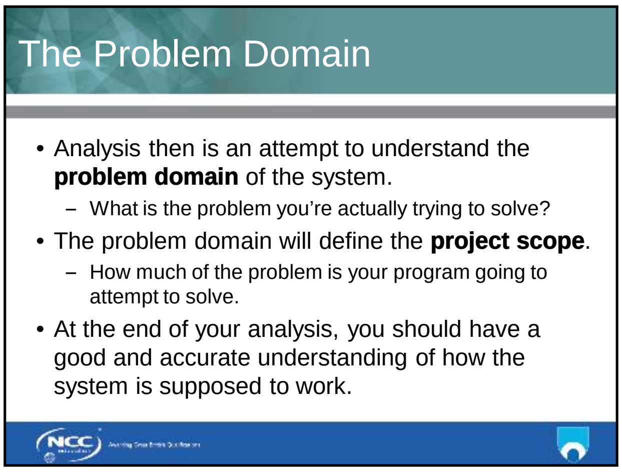
Analysis

- Analysis usually begins from something such as a **problem statement**.
  - A two or three page document which explains what users want.
    - Or rather, what they **think** they want.
- You as an analyst then have to turn that problem statement into something more useful.
  - Through interviews, investigation of existing systems and resolution of ambiguity.



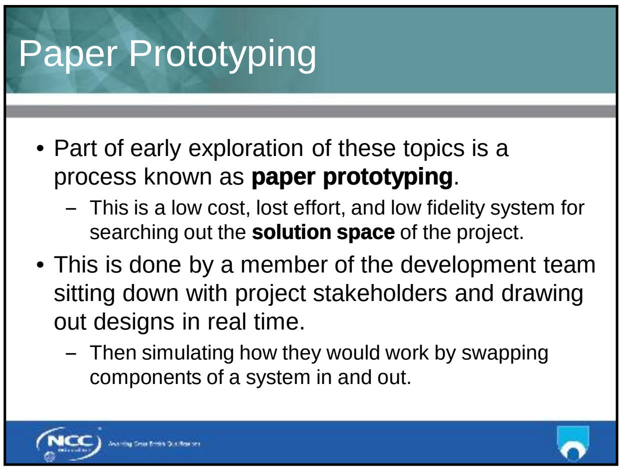
Analysis

- In analysis, we work to build our understanding of the system.
  - We don't focus on how we're going to implement the system later.
- Analysis is the bridge between the information we have and the way the system should actually work.
- Exactly how analysis fits into OOAD is a matter of debate amongst practitioners.



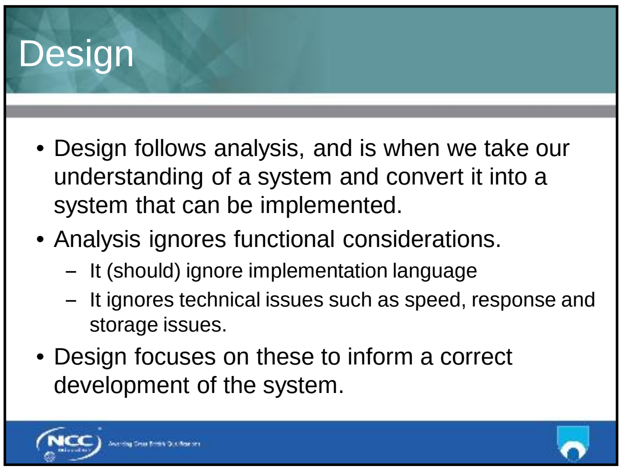
The Problem Domain

- Analysis then is an attempt to understand the **problem domain** of the system.
  - What is the problem you're actually trying to solve?
- The problem domain will define the **project scope**.
  - How much of the problem is your program going to attempt to solve.
- At the end of your analysis, you should have a good and accurate understanding of how the system is supposed to work.



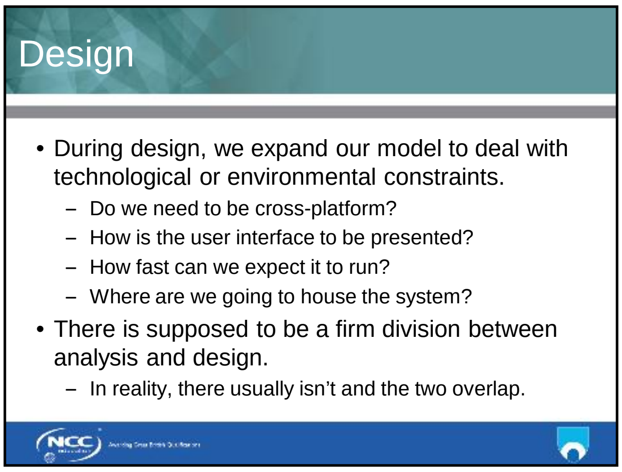
Paper Prototyping

- Part of early exploration of these topics is a process known as **paper prototyping**.
  - This is a low cost, lost effort, and low fidelity system for searching out the **solution space** of the project.
- This is done by a member of the development team sitting down with project stakeholders and drawing out designs in real time.
  - Then simulating how they would work by swapping components of a system in and out.



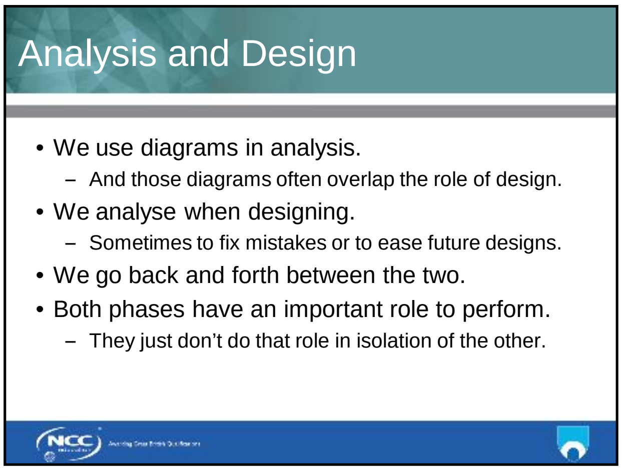
Design

- Design follows analysis, and is when we take our understanding of a system and convert it into a system that can be implemented.
- Analysis ignores functional considerations.
  - It (should) ignore implementation language
  - It ignores technical issues such as speed, response and storage issues.
- Design focuses on these to inform a correct development of the system.



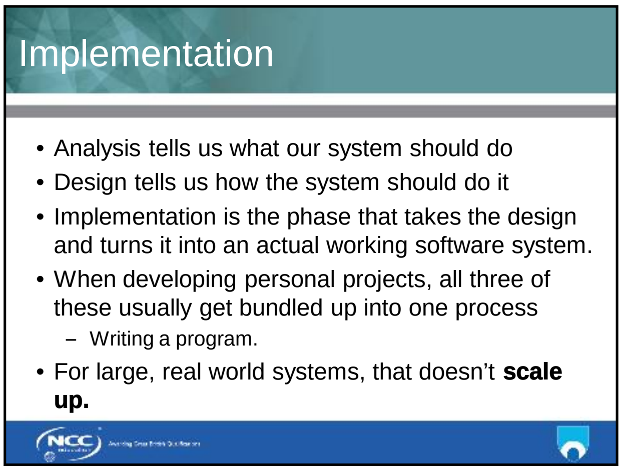
Design

- During design, we expand our model to deal with technological or environmental constraints.
  - Do we need to be cross-platform?
  - How is the user interface to be presented?
  - How fast can we expect it to run?
  - Where are we going to house the system?
- There is supposed to be a firm division between analysis and design.
  - In reality, there usually isn't and the two overlap.



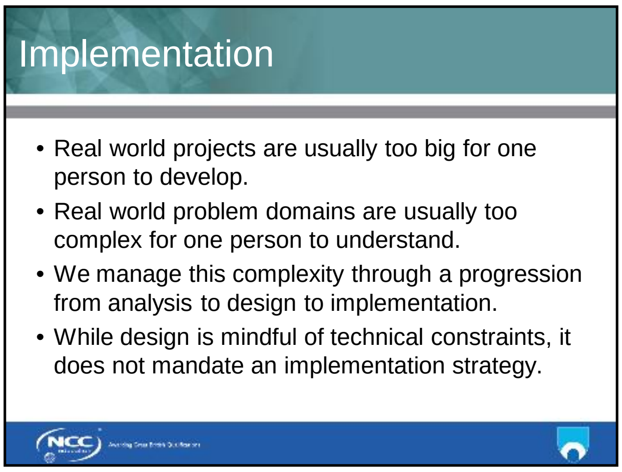
Analysis and Design

- We use diagrams in analysis.
  - And those diagrams often overlap the role of design.
- We analyse when designing.
  - Sometimes to fix mistakes or to ease future designs.
- We go back and forth between the two.
- Both phases have an important role to perform.
  - They just don't do that role in isolation of the other.



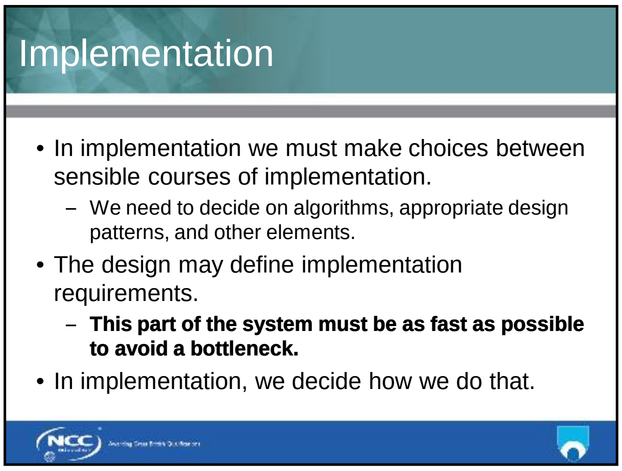
Implementation

- Analysis tells us what our system should do
- Design tells us how the system should do it
- Implementation is the phase that takes the design and turns it into an actual working software system.
- When developing personal projects, all three of these usually get bundled up into one process
  - Writing a program.
- For large, real world systems, that doesn't **scale up.**



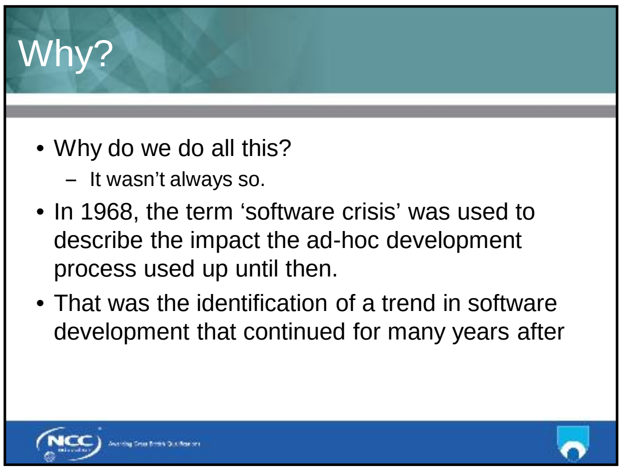
Implementation

- Real world projects are usually too big for one person to develop.
- Real world problem domains are usually too complex for one person to understand.
- We manage this complexity through a progression from analysis to design to implementation.
- While design is mindful of technical constraints, it does not mandate an implementation strategy.



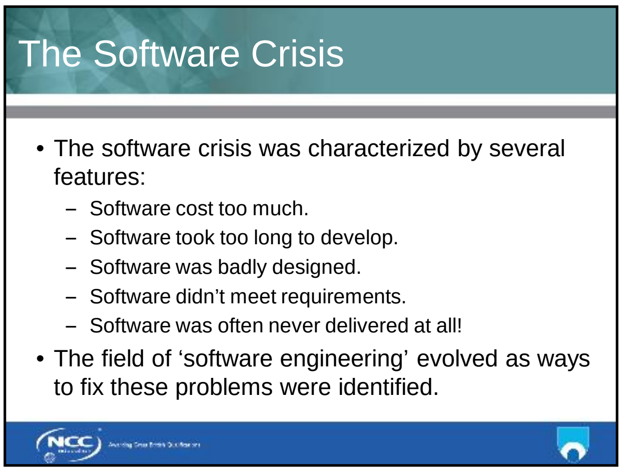
Implementation

- In implementation we must make choices between sensible courses of implementation.
  – We need to decide on algorithms, appropriate design patterns, and other elements.
- The design may define implementation requirements.
  – **This part of the system must be as fast as possible to avoid a bottleneck.**
- In implementation, we decide how we do that.



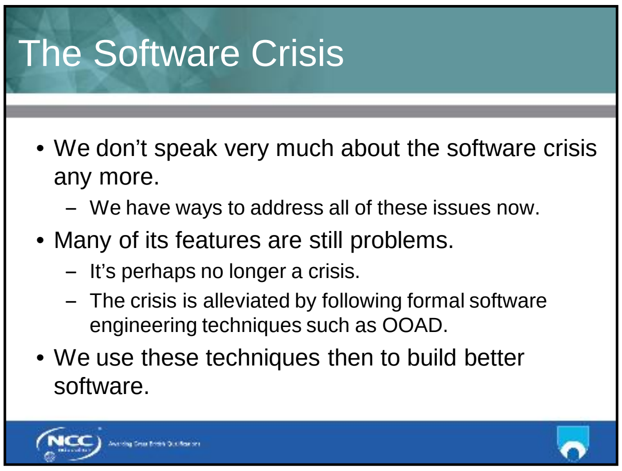
Why?

- Why do we do all this?
  – It wasn't always so.
- In 1968, the term 'software crisis' was used to describe the impact the ad-hoc development process used up until then.
- That was the identification of a trend in software development that continued for many years after



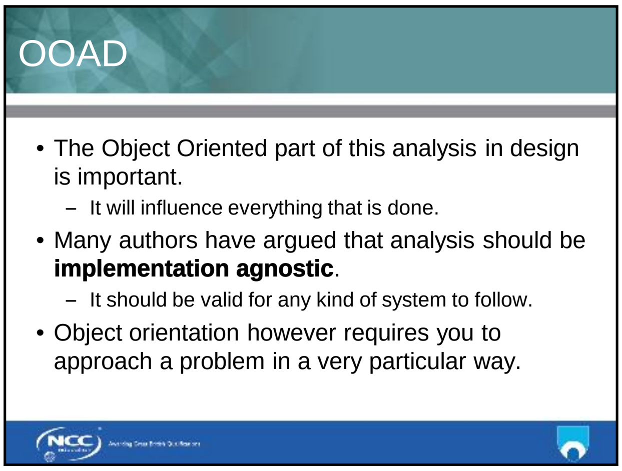
The Software Crisis

- The software crisis was characterized by several features:
  - Software cost too much.
  - Software took too long to develop.
  - Software was badly designed.
  - Software didn't meet requirements.
  - Software was often never delivered at all!
- The field of 'software engineering' evolved as ways to fix these problems were identified.



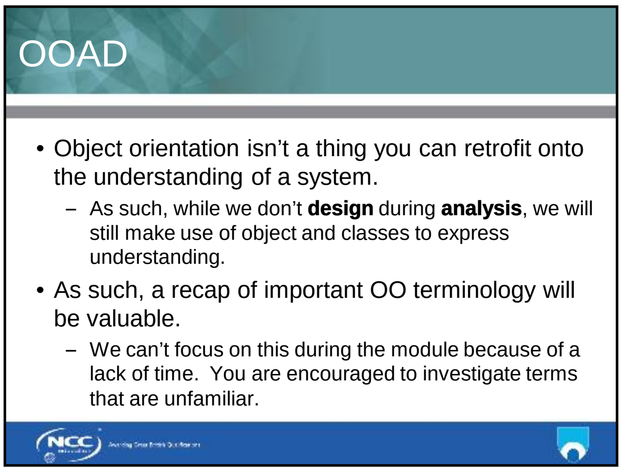
The Software Crisis

- We don't speak very much about the software crisis any more.
  - We have ways to address all of these issues now.
- Many of its features are still problems.
  - It's perhaps no longer a crisis.
  - The crisis is alleviated by following formal software engineering techniques such as OOAD.
- We use these techniques then to build better software.



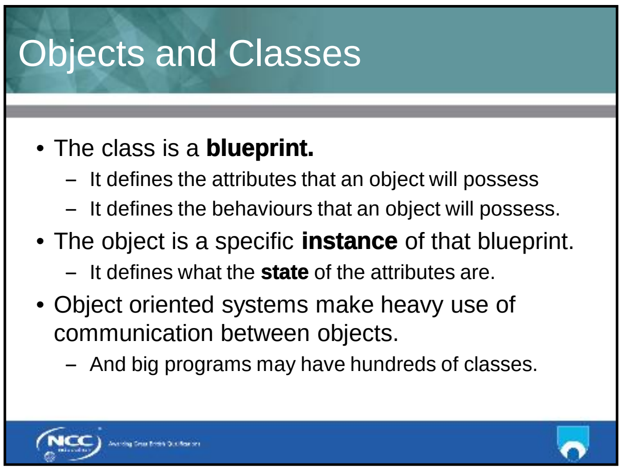
OOAD

- The Object Oriented part of this analysis in design is important.
  - It will influence everything that is done.
- Many authors have argued that analysis should be **implementation agnostic**.
  - It should be valid for any kind of system to follow.
- Object orientation however requires you to approach a problem in a very particular way.



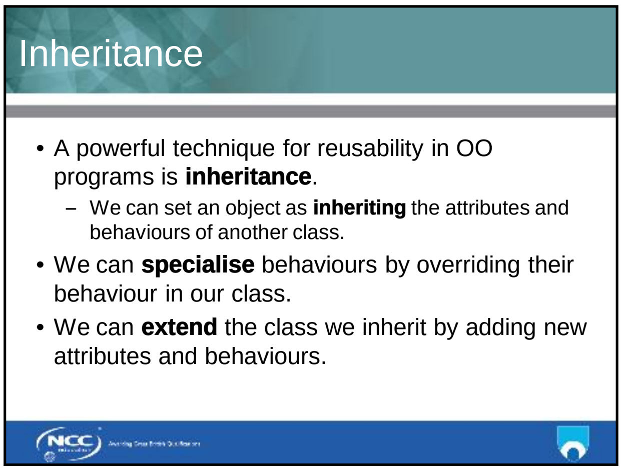
OOAD

- Object orientation isn't a thing you can retrofit onto the understanding of a system.
  - As such, while we don't **design** during **analysis**, we will still make use of object and classes to express understanding.
- As such, a recap of important OO terminology will be valuable.
  - We can't focus on this during the module because of a lack of time. You are encouraged to investigate terms that are unfamiliar.



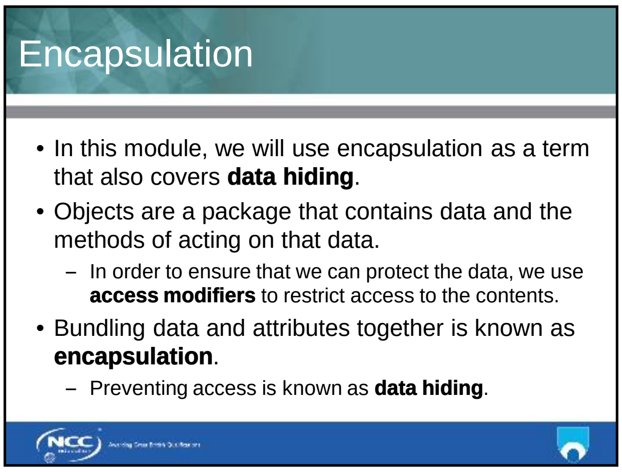
Objects and Classes

- The class is a **blueprint.**
  - It defines the attributes that an object will possess
  - It defines the behaviours that an object will possess.
- The object is a specific **instance** of that blueprint.
  - It defines what the **state** of the attributes are.
- Object oriented systems make heavy use of communication between objects.
  - And big programs may have hundreds of classes.



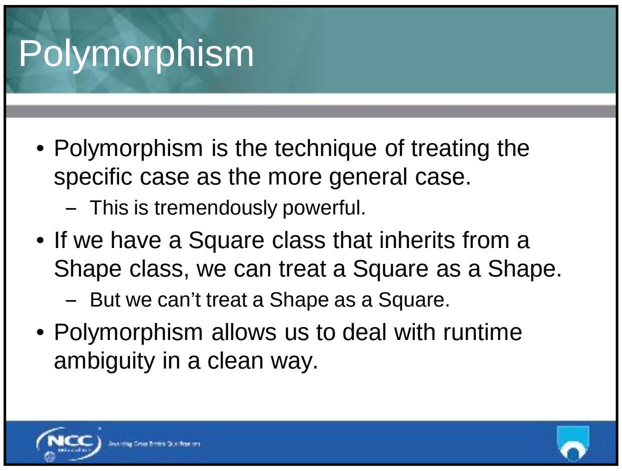
Inheritance

- A powerful technique for reusability in OO programs is **inheritance**.
  - We can set an object as **inheriting** the attributes and behaviours of another class.
- We can **specialise** behaviours by overriding their behaviour in our class.
- We can **extend** the class we inherit by adding new attributes and behaviours.



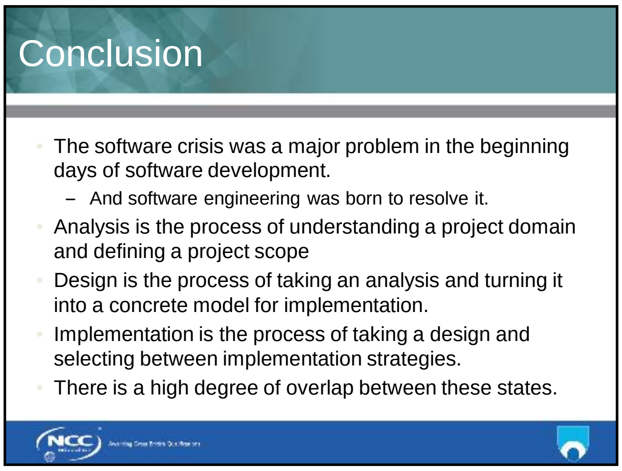
Encapsulation

- In this module, we will use encapsulation as a term that also covers **data hiding**.
- Objects are a package that contains data and the methods of acting on that data.
  - In order to ensure that we can protect the data, we use **access modifiers** to restrict access to the contents.
- Bundling data and attributes together is known as **encapsulation**.
  - Preventing access is known as **data hiding**.



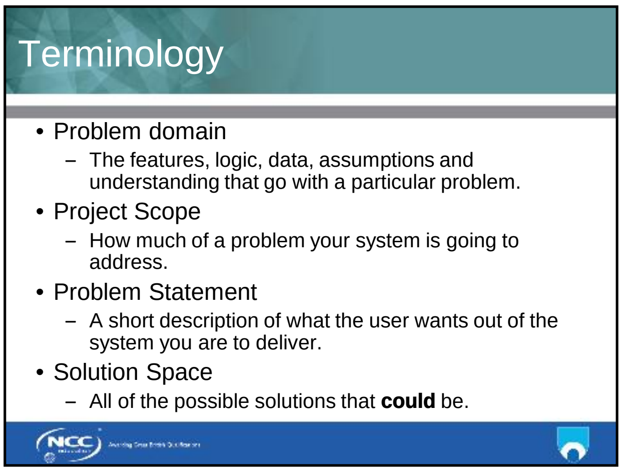
Polymorphism

- Polymorphism is the technique of treating the specific case as the more general case.
  - This is tremendously powerful.
- If we have a Square class that inherits from a Shape class, we can treat a Square as a Shape.
  - But we can't treat a Shape as a Square.
- Polymorphism allows us to deal with runtime ambiguity in a clean way.



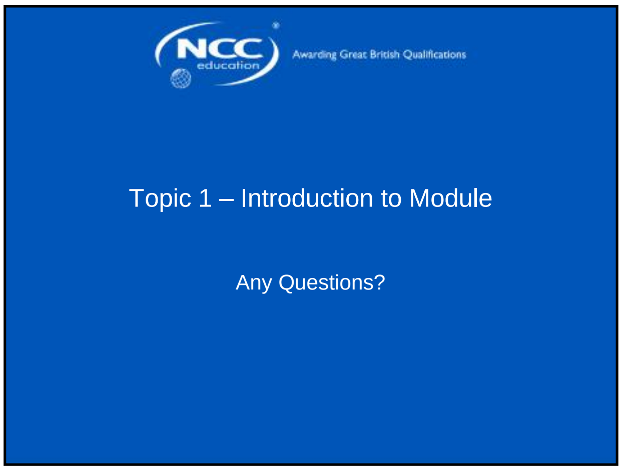
Conclusion

- The software crisis was a major problem in the beginning days of software development.
  - And software engineering was born to resolve it.
- Analysis is the process of understanding a project domain and defining a project scope
- Design is the process of taking an analysis and turning it into a concrete model for implementation.
- Implementation is the process of taking a design and selecting between implementation strategies.
- There is a high degree of overlap between these states.



Terminology

- Problem domain
  - The features, logic, data, assumptions and understanding that go with a particular problem.
- Project Scope
  - How much of a problem your system is going to address.
- Problem Statement
  - A short description of what the user wants out of the system you are to deliver.
- Solution Space
  - All of the possible solutions that **could** be.



Awarding Great British Qualifications

Topic 1 – Introduction to Module

Any Questions?