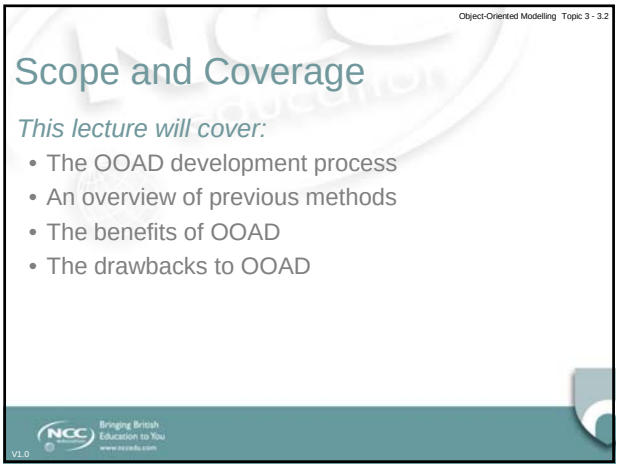
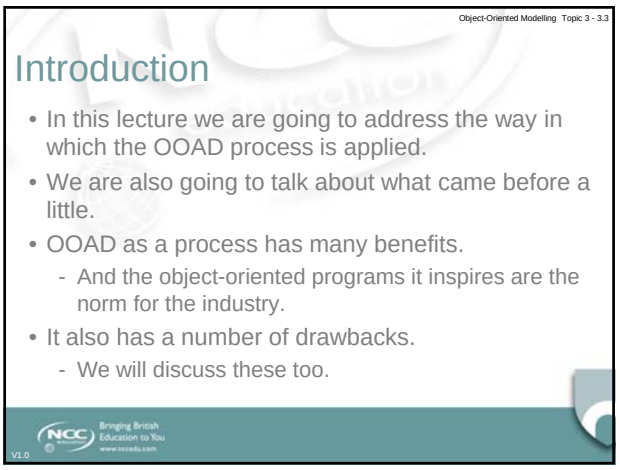

Bringing British
Education to You
www.nccedu.com
Analysis, Design and Implementation
Topic 3:
Object-Oriented Modelling
V1.0


Object-Oriented Modelling Topic 3 - 3.2
Scope and Coverage
This lecture will cover:
• The OOAD development process
• An overview of previous methods
• The benefits of OOAD
• The drawbacks to OOAD
V1.0


Object-Oriented Modelling Topic 3 - 3.3
Introduction
• In this lecture we are going to address the way in which the OOAD process is applied.
• We are also going to talk about what came before a little.
• OOAD as a process has many benefits.
- And the object-oriented programs it inspires are the norm for the industry.
• It also has a number of drawbacks.
- We will discuss these too.
V1.0

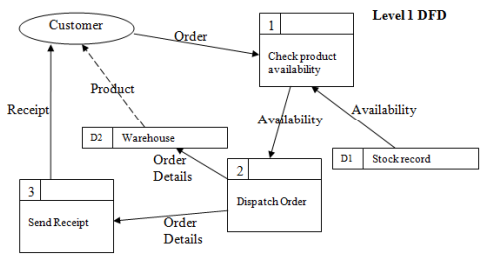
In the Past...

- In the past, most analysis and design progressed through the use of two systems.
  - Structured Systems Analysis and Design (SSADM)
  - The Waterfall Model
    (We touched on this briefly during the first lecture.)
- As software systems grew in size and complexity, these tools ceased to scale up.
- In addition, they were somewhat difficult to change to adapting circumstances.

SSADM

- SSADM stressed a 'data-oriented' approach to developing diagrams for systems.
  - Most of the diagramming was done through ***data flow diagrams*** (DFDs)
- These emphasised the way in which data was stored, manipulated and passed through the system.
- It was a useful notation, but as systems grew so too did the diagrams.

Example DFD



Data Flow

- Each of the boxes in the level one DFD would be expanded into a DFD of its own, and so on.
- Rapidly, it became difficult to explore a complex program while also understanding the context in which parts functioned.
- Because of the focus on data, documents had to be maintained defining what data was and where it was used.

Data Dictionaries

- These were called data dictionaries, and it was not uncommon for them to run to hundreds of pages.
- They stored details of formats, defaults, validation routines, and so forth.
- Any time data was used, its use was recorded in the dictionary.
- Any time new data was generated, it was recorded in the dictionary.
- It was very unwieldy.

Software Complexity

- As the complexity of software increases, these diagrams became more cumbersome.
- Object-oriented analysis and design was introduced to help simplify the architecture of large, complex programs.
  - An object is a small, self-contained program of its own.
  - The system is the interaction of all the objects in a program.
- This allows for more compact representation in diagrams.

Object-Orientation

- Object-orientation is a progression from the procedural programming paradigm of earlier languages.
  - Objects add an extra level of modularity on top of the existing functions permitted.
- Programs written using structured programming often lacked maintainability.
- Object-orientation was developed to address this deficiency.

Benefits of OOAD

Object-oriented analysis and design has a number of advantages over structured analysis and design:

- Systems are more effectively decomposed into units.
- Good OOAD results in components that are more easily maintained.
- Good OOAD results in components that can be more easily reused between systems.
- OOAD more naturally models how systems work in practise.
  - Data flow analysis is an artificial, algorithmic overview.

Drawbacks of OOAD

There are drawbacks too:

- Large systems will have hundreds of classes, and interactions can be complicated.
- It is easy to badly design classes.
- Object-orientation requires a trade-off between coupling and cohesion - you cannot have it all.
- While it more naturally models how systems work, it is still an unusual way for people to think.

A Simple OOAD Process - 1

- Much of the benefit of OOAD can be obtained through the use of a five step process.
  - ***Identify the needs of users***
    Documented via use case diagrams
  - ***Detail the steps needed for each of the requirements***
    Done through activity diagrams
  - ***Decompose the requirements for the system***
    Break it down into components via class diagrams
  - ***Define out the interactions***
    Bring it all together in a component diagram
  - ***Iterate***

A Simple OOAD Process - 2

- Iteration is an important part of OOAD.
  - You will never get it right the first time.
  - New requirements and information will be introduced all the time.
- Incremental analysis and design is simplest.
  - Do not try to solve the whole problem at once.
  - Pick a starting point, and work from that.
- Good design is user centric.
  - You need to know what the users have to say.

Decomposition

- Understanding any complex system is an exercise in ***decomposition***.
  - You must be able to partition the whole into manageable subsections.
- Abstraction is an important part of this process.
  - You need to be able to view the different parts at a suitable level of granularity.
- Incremental development is the process of successively refining your abstractions.

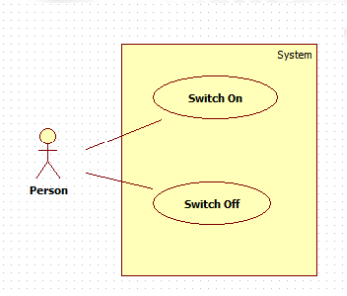
The Use Case Diagram - 1

- The ***Use case diagram*** is an important tool in managing your abstractions. It allows you to represent the broad interactions between parts of a system.
- It is used to represent the set of functionality that must be supported for each part. Those parts are called ***actors***.
  - They may be users.
  - They may be subsystems.

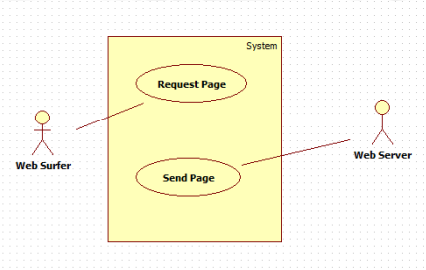
The Use Case Diagram - 2

- Use case diagrams do not show interactions between actors. That is beyond the scope of our analysis and design.
- Actors are represented by stick figures.
- Actions are represented by ovals in which a broad description of the process is placed.
- A specific interaction is defined as a line which connects the actor and the action they can perform.

Use Case Diagram Example 1



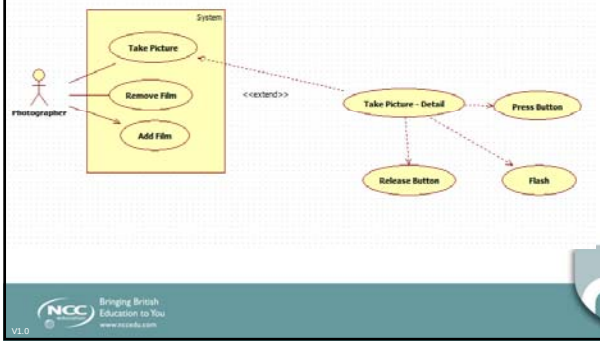
Use Case Diagram Example 2



Use Case Diagrams

- Use case diagrams are supposed to show only broad strokes of interaction.
- However, sometimes we want to specialise a specific action if it has clearly defined subtasks.
- To do this we create a separate diagram and flesh out the interaction.
  - We can make an interaction have multiple parts, providing a <<uses>> line to indicate subtasks.

Use Case Diagram Example 3



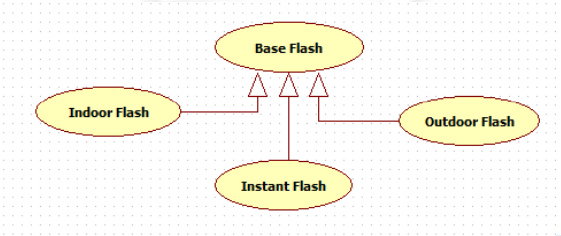
Use Case Diagrams - 2

- Note that no order is imposed in use diagrams. We will handle that in a different, later diagram.
- You can think of this as a high level overview of your user interface.
  - You need to permit ways for people to do all of the things you have indicated on the diagram.
- Generating the use case diagram will be a result of interaction with the users and the problem statement.

Use Case Diagrams - 3

- A third special syntax of a use case diagram permits you to indicate that one kind of action derives from another.
- This is the ***extends*** syntax, and is used to demonstrate both inheritance and polymorphism in a diagram.
  - You will not have to do this until quite late into the OOAD process.

Use Case Diagram Example 4



Identifying Use Cases - 1

- We can use a technique called ***event decomposition*** to arrive at a list of candidate events for our system.
  - We treat the system as a black box.
  - We focus on the things that happen to the black box.
- We may end up discarding or combining the events that we come up with.
  - That is all part of the iterative process.
  - What we need to begin with is a starting point.

Identifying Use Cases - 2

- There are three main kinds of event we need to look at.
  - External events
  - Temporal events
  - State events
- We consider each of these events in relation to the potential actor.
  - This technique is merely a way of focusing our thinking.
  - The starting point will still be the problem statement or the users.

External Events

- ***External events*** exist outside the system, and are usually initiated by a third party outside the scope of our system, for example, the customer of a web page, or the database administrator.
- We then document all of the potential interactions that each of these actors may be required to perform. Each of these then becomes a candidate for a use case diagram.

Example Candidates

Customer	**List products by category** **List products by price** **Buy a product** **Read reviews of a product** **View shopping basket** **Update account details**
Administrator	Add new products Remove products Modify products Modify customer details

Temporal Events - 1

- ***Temporal events*** are those that occur as a result of reaching a particular point in time, for example 'End of the month, so handle salaries.'
- Sometimes these events will be triggered by external entities. A user may set up an report that should be mailed to them every week.
- We determine temporal events by detailing any specific deadlines or recurring functionality.

Temporal Events - 2

- Temporal events do not necessarily occur at a fixed time. They may instead occur after time passed.
  - E.g. Debit the customer's account ten minutes after they have purchased an item.
- The occurrence of the timed event is the temporal aspect. Setting the event to occur is often an external event.

State Events

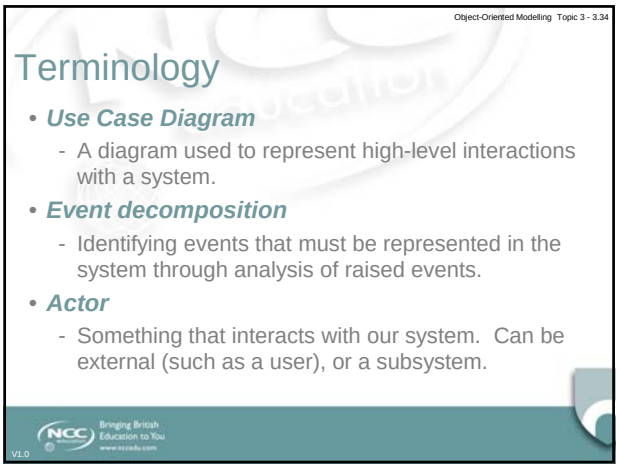
- ***State events*** are those that occur when the data in a system reaches a point where processing is required.
  - E.g. When stock drops below a certain amount, email the procurement department.
- Normally these occur as a result of other events.
  - Temporal or state
  - A customer buys a product, which adjusts the stock, which throws up a state event.

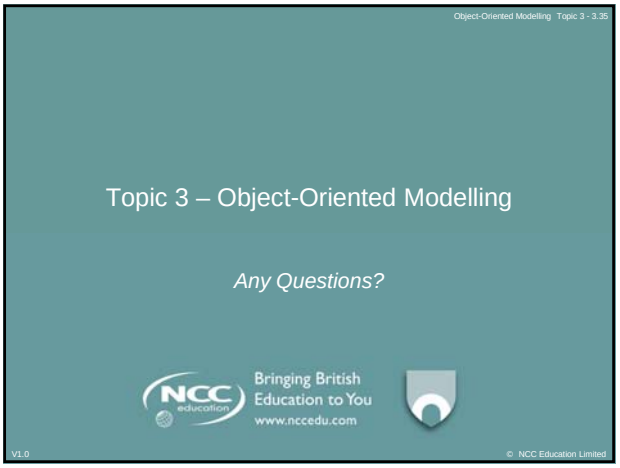
Choosing between Events

- What we get out of this is a list of ***candidates***. They are not all going to be worthwhile.
- The only ones we care about are those that directly affect our system.
- We do not care about the events that lead up to the interaction, or those that follow them.
- We need to strive for a consistent level of detail across the events. This may involve breaking some out into multiple events, or combining others.

Conclusion

- OOAD is an evolution from structured analysis and design.
- It stresses interaction of components rather than the flow of data between algorithms.
- Use case diagrams are used to represent a high level view of actor interactions.
- There are many ways to develop use case diagrams.
  - Event decomposition can be a useful technique.


Object-Oriented Modelling Topic 3 - 3.34
Terminology
• Use Case Diagram
- A diagram used to represent high-level interactions with a system.
• Event decomposition
- Identifying events that must be represented in the system through analysis of raised events.
• Actor
- Something that interacts with our system. Can be external (such as a user), or a subsystem.
Bringing British Education to You
www.nccedu.com
V1.0


Object-Oriented Modelling Topic 3 - 3.35
Topic 3 – Object-Oriented Modelling
Any Questions?
Bringing British Education to You
www.nccedu.com
V1.0
© NCC Education Limited