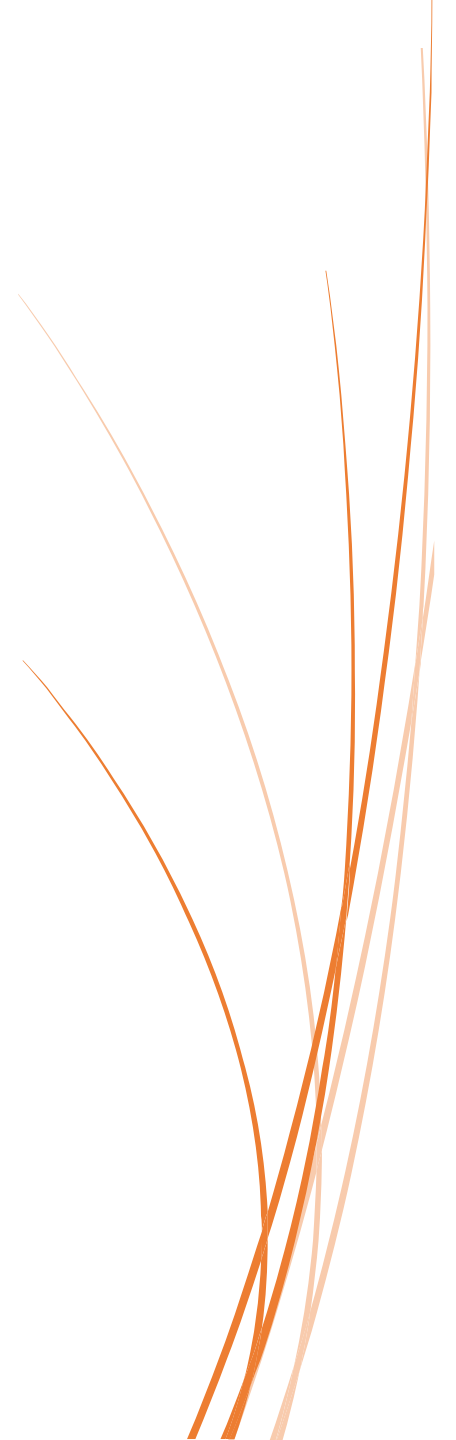


USDP

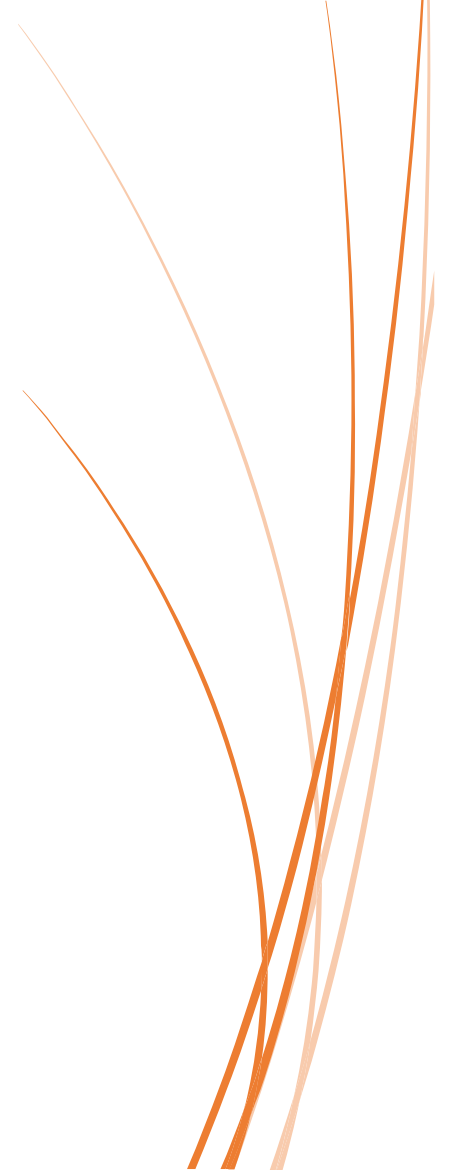
Use case and activity diagrams;
Analysis diagrams



In this lecture

We will:


- Introduce the USDP
- Introduce UML diagrams
 - Use case diagrams
 - Activity diagrams
 - Analysis diagrams




UML is only a language

- The UML is a language, not a design process
 - Use it to describe the results of designing a solution to a problem
- A software design process provides guidance on how to...
 - Describe a problem
 - Move from problem to solution
 - Encode the solution as a computer program

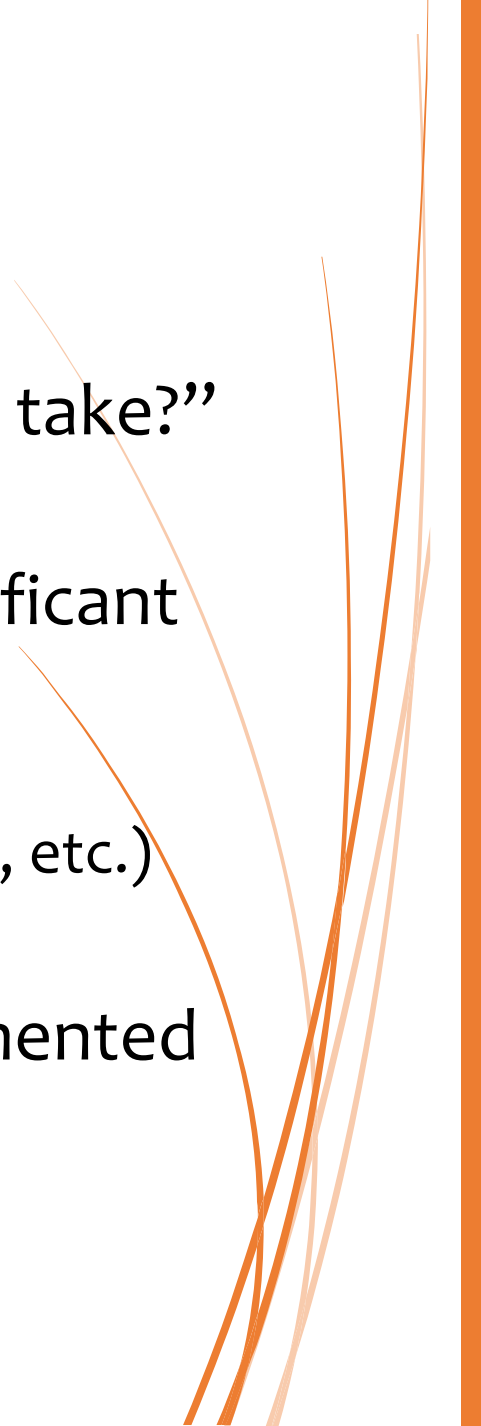
USDP

- The Unified Software Development Process (Jacobson et al, 1999)
 - Grew out of thirty years of development and experience
 - The USDP is a set of activities that transforms user requirements into software
 - Uses the UML to describe the results of the process
 - The USDP is:
 - Use-case driven
 - Architecture-centric
 - Iterative and incremental
- 

USDP is use-case driven

- A use case model answers the question, “What does the user do with the system?”
 - Use cases drive the entire development process
 - The design process proposes a solution to satisfy the use case model
 - The implementation builds the proposed solution one use case at a time
 - The testing process verifies that the implementation satisfies the use case model
- 

USDP is architecture-centric


- A software architecture answers the question, “What form does the system take?”
 - Software architecture defines the significant aspects of the application
 - Static – structure (i.e. key classes, etc.)
 - Dynamic – behaviour (i.e. key interactions, etc.)
 - The application is designed and implemented around the architecture
- 

Use cases and architecture

- Use cases and architecture are developed together
 - They mature side-by-side over time
- Every product has two features
 - Function – in USDP, defined by the use cases
 - Form – in USDP, defined by the architecture



Iterative and incremental

- The USDP is **iterative**
 - Each iteration addresses a sub-set of the use case model
 - The USDP is **incremental**
 - Each iteration adds to previous work done
 - Each iteration produces a working system
 - Each iteration adds an increment until the system is finished
 - Each iteration is a mini-project
 - It follows a complete workflow – requirements, analysis, design, implementation and testing
 - i.e. it is not hacking!
- 

Our use of USDP & UML

- We shall develop the following diagrams in the order given

- Use case diagram
 - One for the entire application
- Activity diagrams
 - One per use case

Describe the problem domain

- Analysis diagrams
 - One per use case

A step towards the solution

- Class diagram
 - One for the entire application
- Sequence diagrams
 - One per use case

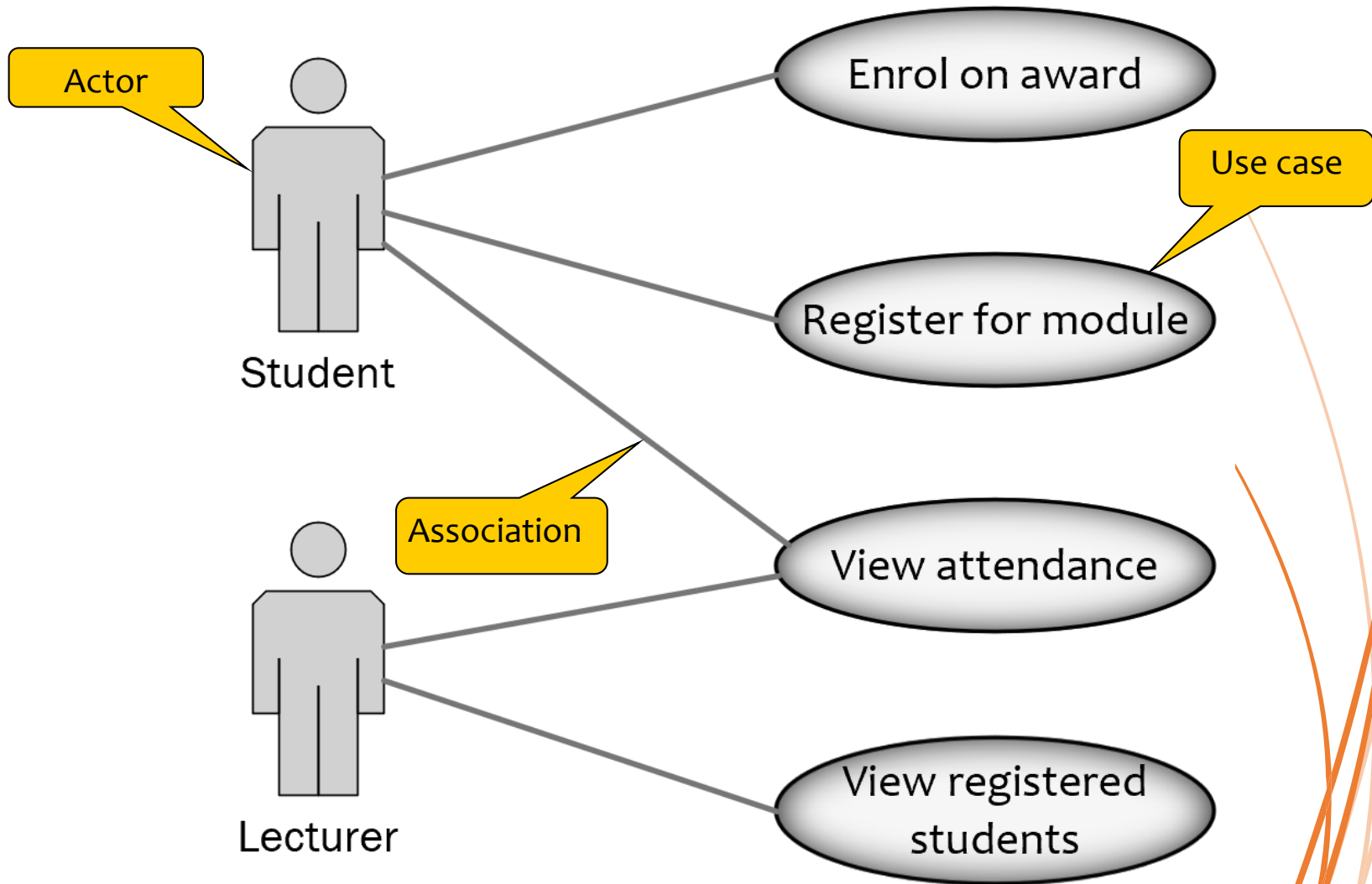
Describe the solution domain

Use Case diagrams

- Actors are entities that interact with the system, and can be
 - People
 - Roles that people play
 - Other systems
- Use cases describe *what* actors want the system to do for them
- A use case is a complete flow of activity
 - Described from the actor's point of view
 - Provides something useful to the actor

Example use cases

The name of a use case completes the sentence: “The actor wants to...”

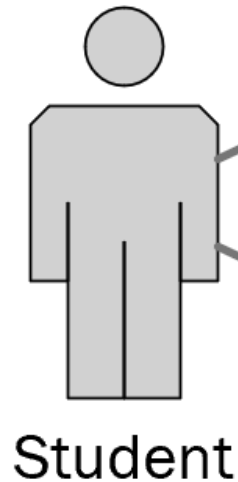


Use case relationships

- <<include>> relationship
 - Shows another use case that **must** be executed at the same time as this one
- <<extend>> relationship
 - Shows another use case **might** be executed at the same time as this one

Example <<include>>

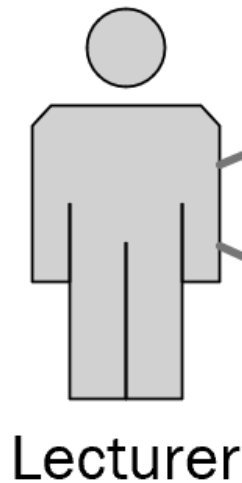
When the “Register for module” use case is executed, “Refresh timetable” **must** execute, too



The “Refresh timetable” use case can execute on its own

Example <<extend>>

When the “View registered students” use case is executed, “View attendance” **might** execute, too



View attendance

<<extend>>

View registered students

This use case can execute on its own

<<include>> vs. <<extend>>

	<<include>>	<<extend>>
Is this use case optional?	No	Yes
Is this use case complete without the other use case?	No	Yes

Beware!

- <<include>> and <<extend>> use cases must ***always*** be complete flows of activity in their own right

Use cases ***are not*** a form of functional decomposition (step-wise refinement)

Describing a use case

- A use case diagram is very abstract
 - i.e. not much detail
- More detail can be provided with
 - Textual descriptions
 - UML diagrams (e.g. activity diagram)
- Textual descriptions
 - Complex workflows can be difficult to follow in text
 - Diagrams can convey complexity better than text

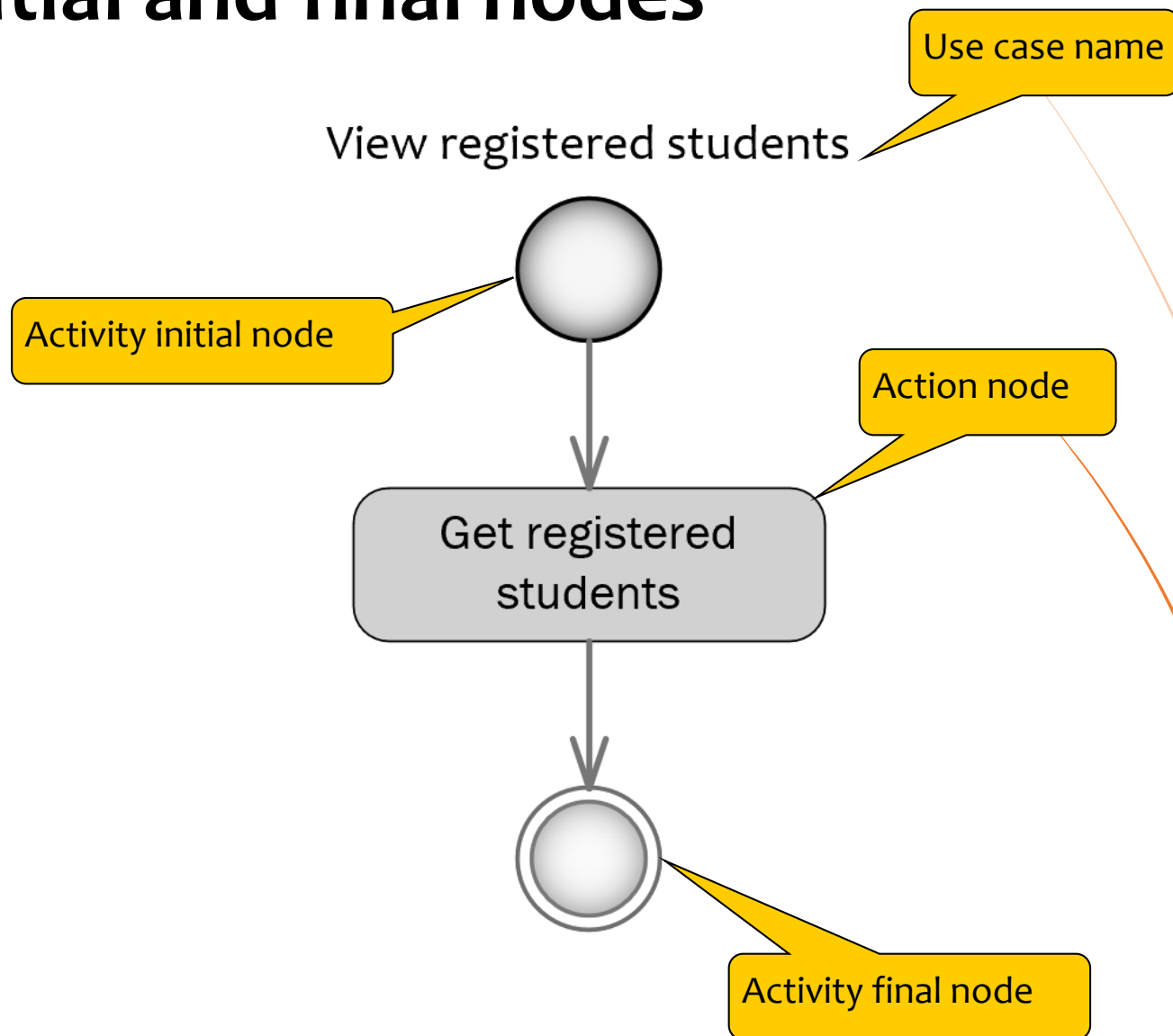
Activity diagrams

- An activity diagram describes the workflow of a single use case
- An activity diagram focuses on:
 - The actions performed when the use case is executed
 - The order in which those activities are done
- Hint
 - When identifying the actions, assume they are done without a computer
 - This will keep you in the problem domain

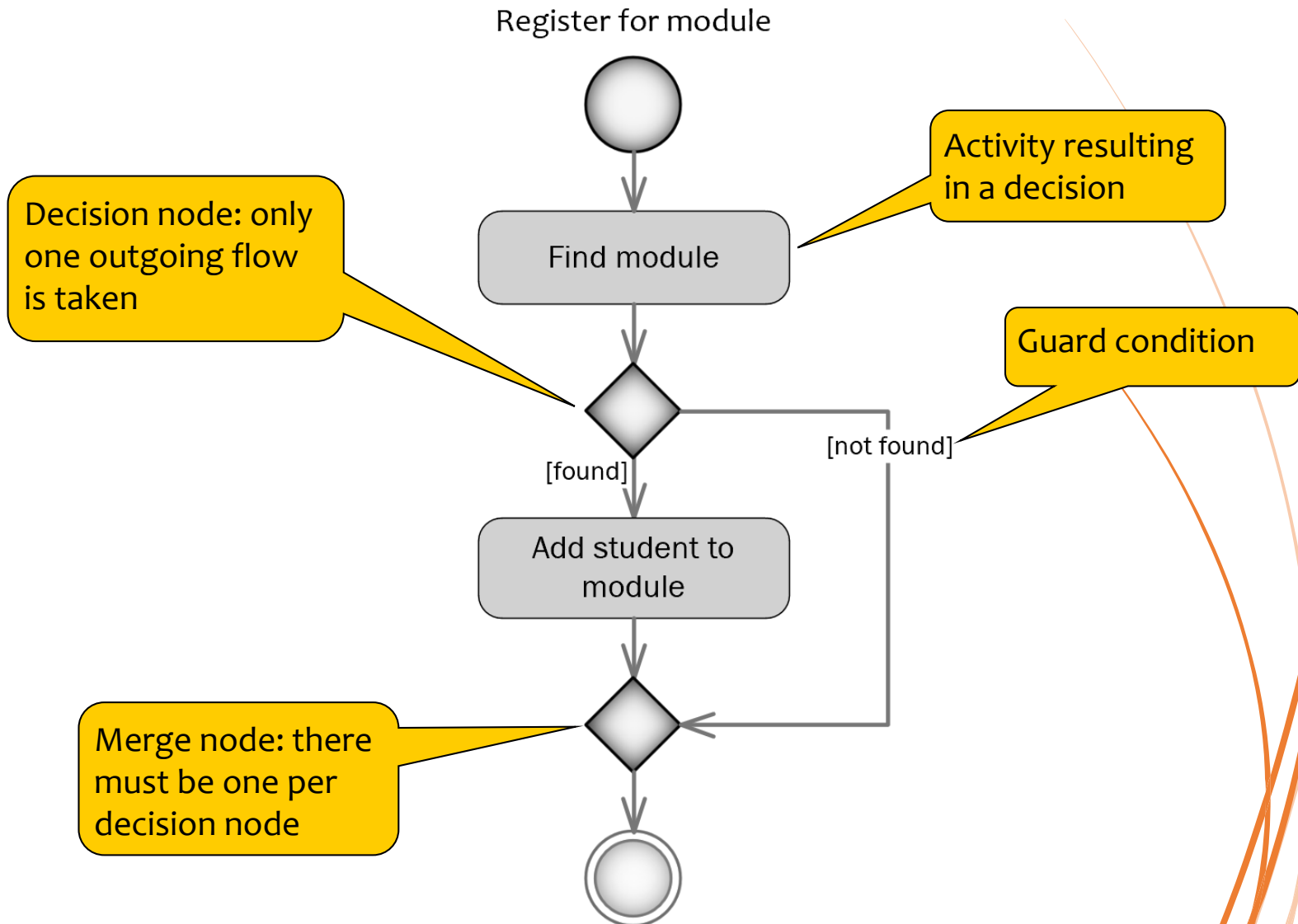
Activity diagrams

- An activity diagram has:
 - Action nodes
 - Control nodes
- An activity has one or more action nodes
 - An action node has exactly one flow in and one flow out
- Control nodes can be:
 - Activity initial and activity final
 - Decision and merge

Initial and final nodes



Decision and merge nodes



Our use of USDP & UML

- Having described the problem domain, we now take a step towards the solution domain

- Use case diagram
 - One for the entire application
- Activity diagrams
 - One per use case

Describe the problem domain

- Analysis diagrams
 - One per use case

A step towards the solution

- Class diagram
 - One for the entire application
- Sequence diagrams
 - One per use case

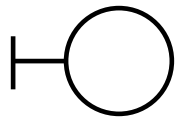
Describe the solution domain

Analysis diagram

- Begin thinking about the types of class that are needed in the solution
 - Without class details like attributes and methods
- One analysis diagram for each use case
 - Identify the analysis classes needed for the use case
 - Identify the messages that are passed between the analysis classes

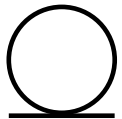
Analysis classes

- Three stereotypes for analysis classes in UML:



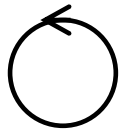
- Boundary class

- Receives messages from the actor
- Communicates with entity and control classes



- Entity class

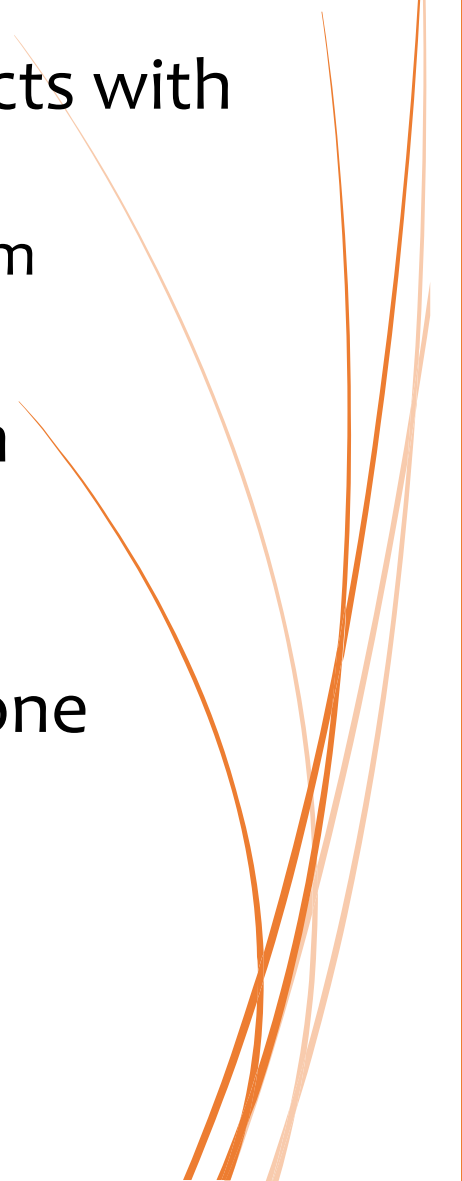
- Stores data in the application



- Control class

- Manages collections of entity classes

Analysis class: Boundary


- The gateway by which the actor interacts with the system
 - e.g. messages passed from actor to system
 - Every boundary class is associated with at least one actor
 - Every actor is associated with at least one boundary class
- 
- The slide features three decorative orange curved lines on the right side, starting from the bottom and curving upwards and outwards. There is also a solid orange vertical bar along the far right edge of the slide.

Analysis class: Entity

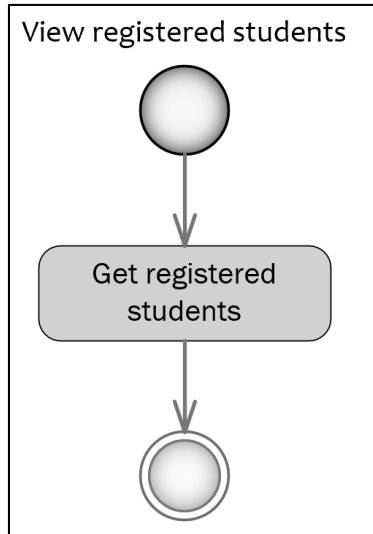
- Stores data that persists over time
within the application
- Models relevant objects or events
within the application
- Not just data; include behaviour as well
 - Reminder: behaviour is represented as responsibilities



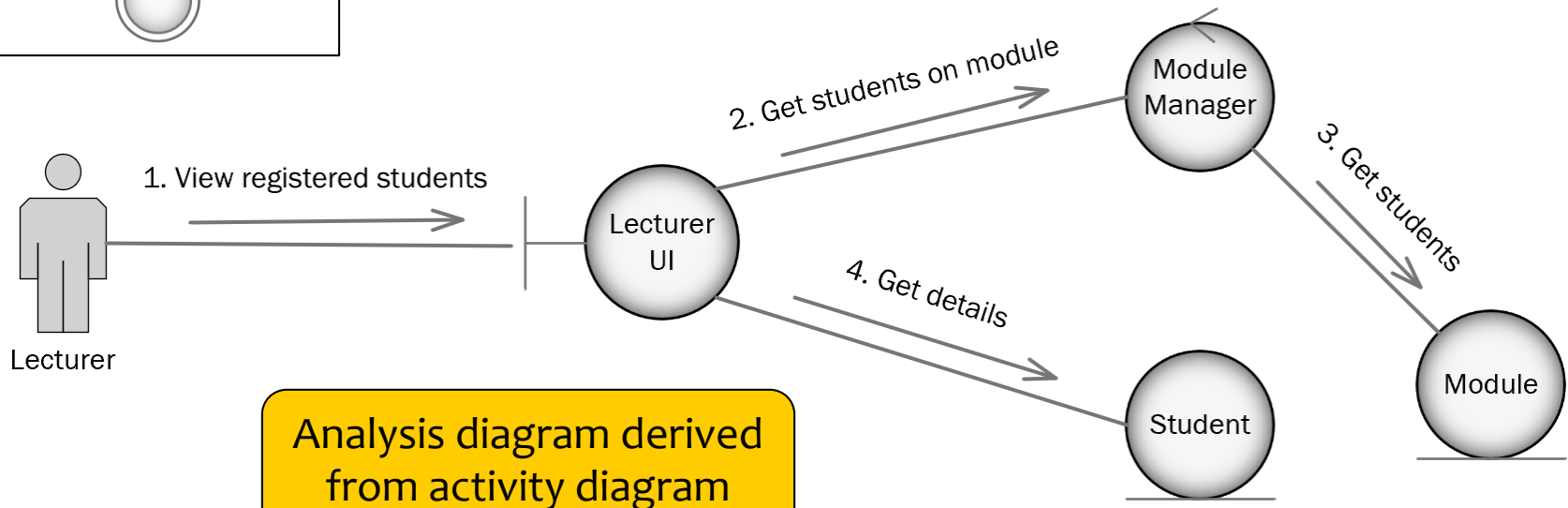
Analysis class: Control

- Manages collections of one or more entity classes
 - Manages complex business logic that involves several other objects
 - Coordinates operations
 - Sequences events
 - Passes messages to other objects
 - Delegates tasks to other objects
- 
- A decorative graphic consisting of several thin, curved orange lines that sweep upwards from the bottom right corner towards the top right of the slide. There is also a solid orange vertical bar along the far right edge of the slide.

Example 1: Analysis diagram

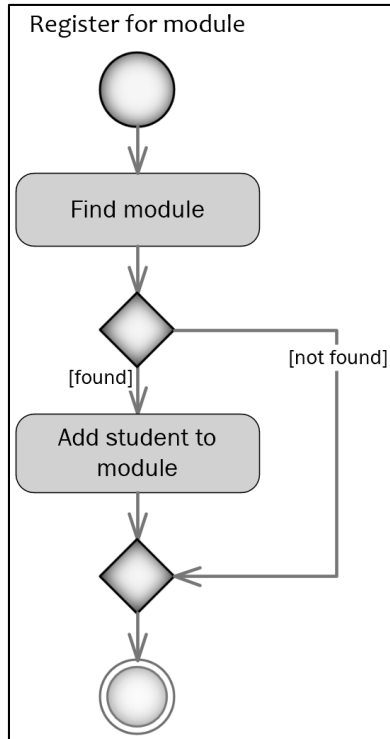


Activity diagram
from earlier



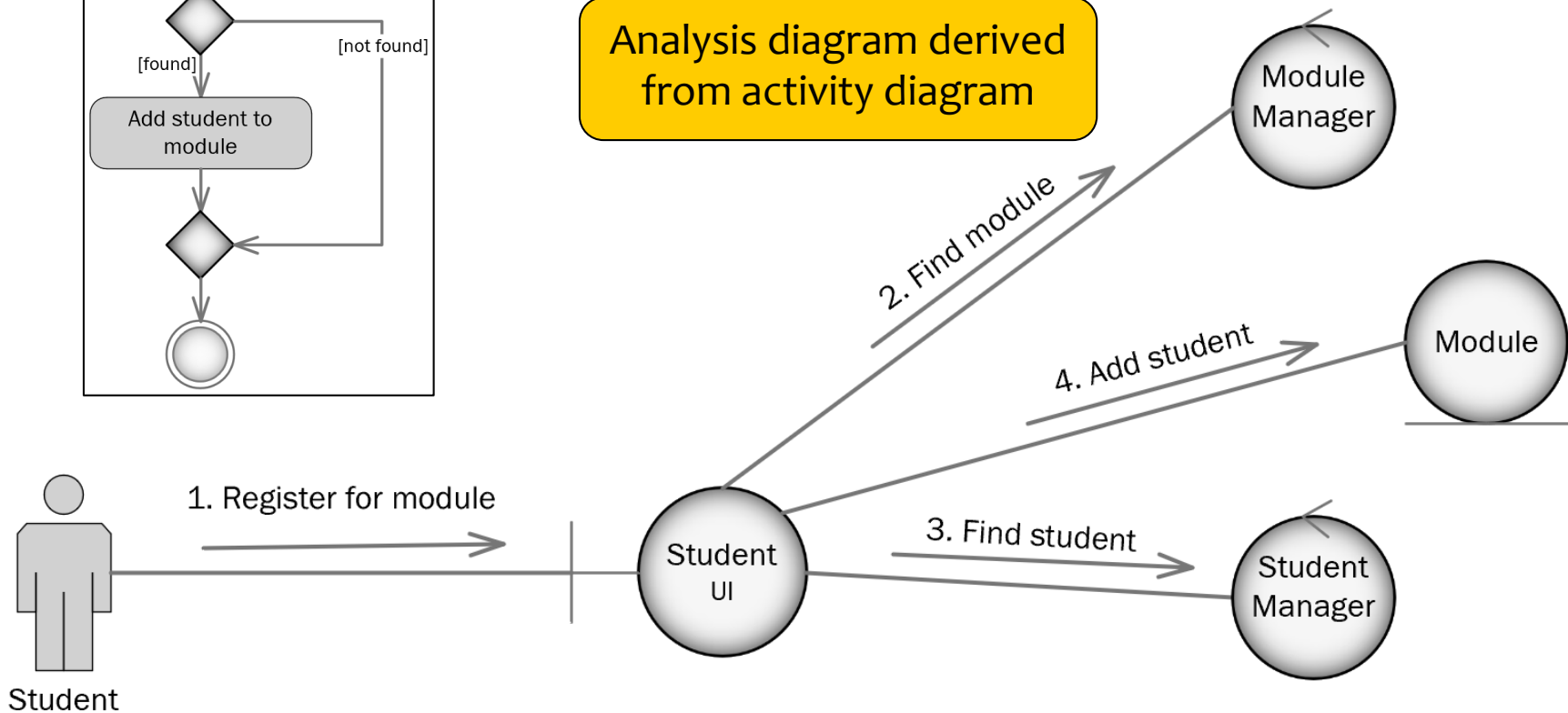
Analysis diagram derived
from activity diagram

Example 2: Analysis diagram

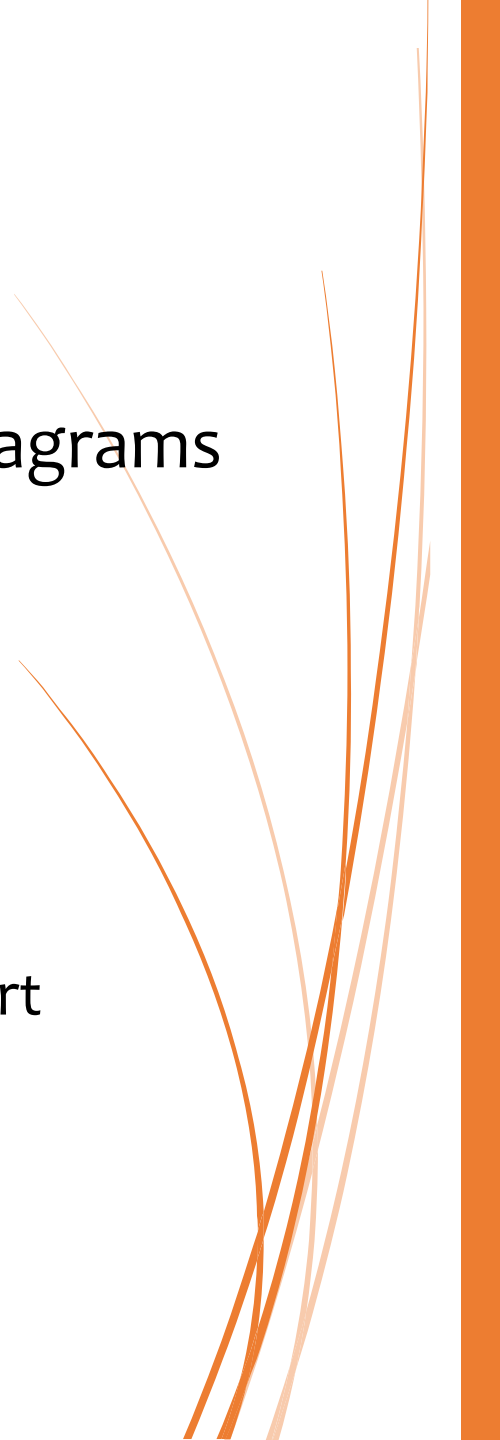


Activity diagram
from earlier

Analysis diagram derived
from activity diagram



You should be able to...

- Describe the USDP
 - Describe and use the following UML diagrams
 - Use case diagrams
 - Activity diagrams
 - Analysis diagrams
 - In the next lecture we will:
 - Introduce the diagrams for the second part of the USDP
 - Analysis diagrams
 - Class diagram
 - Sequence diagrams
- 

References

- Booch, G.; Maksimichuk, R.A.; Engle, M.W.; Young, B.J.; Conallen, J.; Houston, K.A. (2011) *Object-oriented analysis and design with applications*. Addison-Wesley