JACOBS UNIVERSITY

350102 GENERAL INFORMATION & COMMUNICATION TECHNOLOGY II (GENICT)

- DESCRIBING SOFTWARE -

Instructor:	Peter Baumann
email:	p.baumann@jacobs-university.de
tel:	-3178
office:	room 88, Research 1

The Software Life Cycle

Requirements Engineering

↳ Design

↳ Coding

Configuration, Release, & Dependency Management

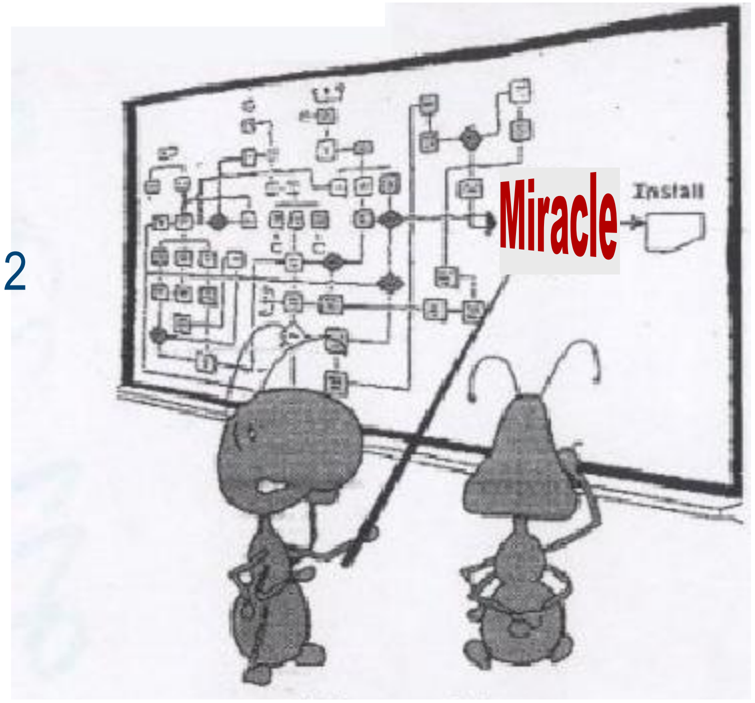
↳ Verification & Testing

↳ Deployment, Maintenance

"Plan? Who needs a plan?" Introduction to UML

based on:
Introduction to the Unified Modeling Language, Chapter 2
Terry Quatrani, UML Evangelist, IBM

Instructor:	Peter Baumann
email:	p.baumann@jacobs-university.de
tel:	-3178
office:	room 88, Research 1



Excellent work! But maybe we should get a little more detailed here...?

What is UML?

- What is UML?
  - "The UML (Unified Modeling Language) is the [OMG] standard language for specifying, visualizing, constructing, and documenting all the artifacts of a software system."
  - Synthesis of notations by Grady Booch, Jim Rumbaugh, Ivar Jacobson, and many others
    - *Rational, Objectory, et al, ...now IBM*
- diagram perspectives
  - Conceptual, specification, implementation

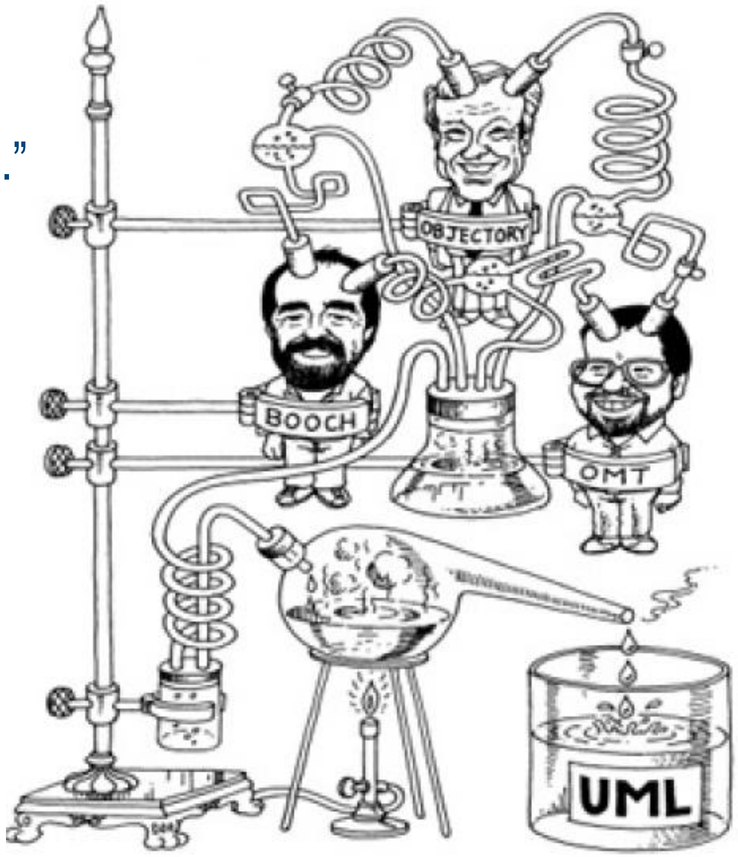


Diagram Types Overview

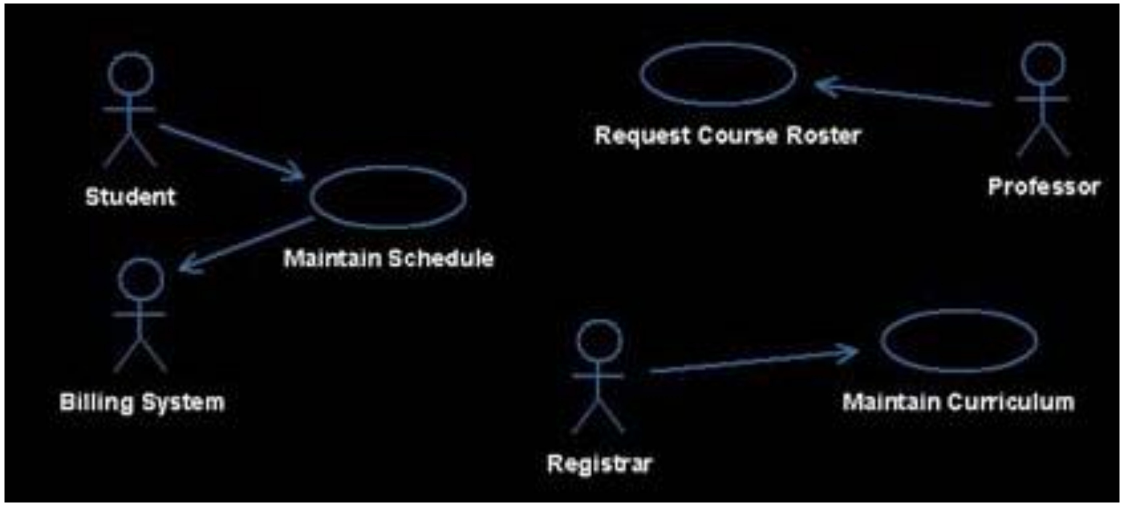
- Main diagram types, according to „80/20 rule“:
  - Use Case Diagram (functional)
  - Activity / Action Diagram (behavioral)
  - Class Diagram (structural)
  - State Diagram (behavioral)
  - Sequence Diagram (behavioral)

1,000+ pages

- Further, not addressed here:
  - Object Diagram (structural), Collaboration Diagram (structural), Package Diagram (structural), Deployment Diagram (structural)
  - Interaction Diagram ::= Collaboration Diagram | Sequence Diagram

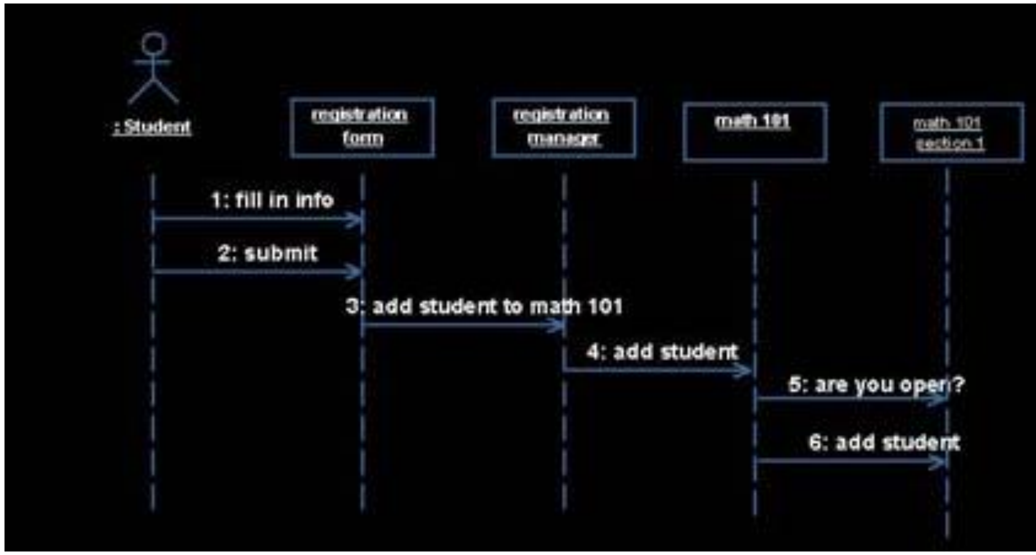
Use Case Diagrams

- use case = chunk of functionality, not a software module
  - Should contain a verb in its name
- actor = someone or some thing interacting with system under development
  - Aka role in scenario
- Visualize relationships between actors and use cases
- capture high-level alternate scenarios, get customer agreement (early !)



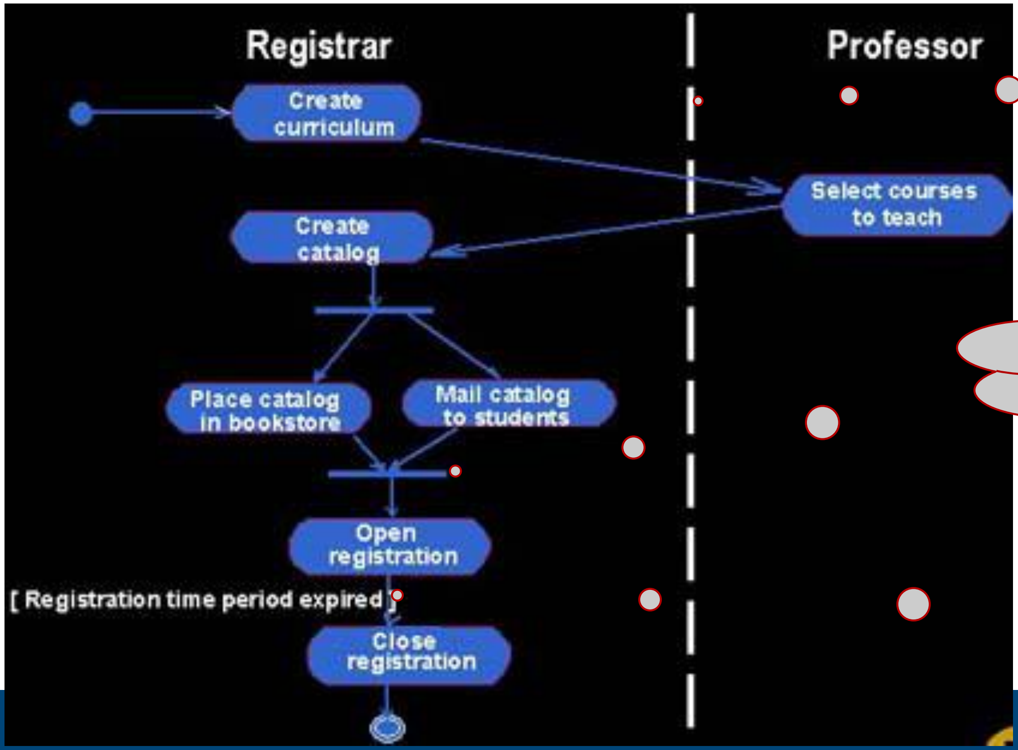
Sequence Diagrams

- Displays object interactions arranged in a time sequence
- Can be from user's perspective!
  - good for: showing what's going on and driving out requirements when interacting with customers
- How many SDs? Rule of thumb:
  - for every basic flow of every use case
  - for high-level, risky scenarios
- Useful for designer and customer to answer the question:
  „what objects and interactions will I need to accomplish the functionality specified by the flow of events?"



Activity Diagrams

- Represents the overall flow of control
- Graphical workflow of activities and actions
  - like flow chart, but user-perceived actions (business model)



Swimlanes

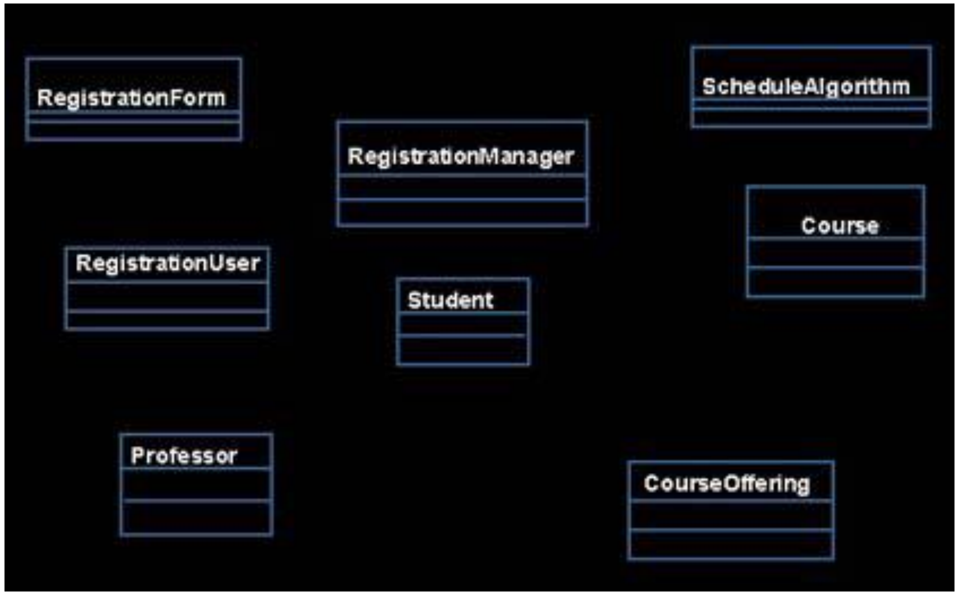
Synchronisation bar (fork/join)

Transition guard

Class Diagrams

JACOBS UNIVERSITY

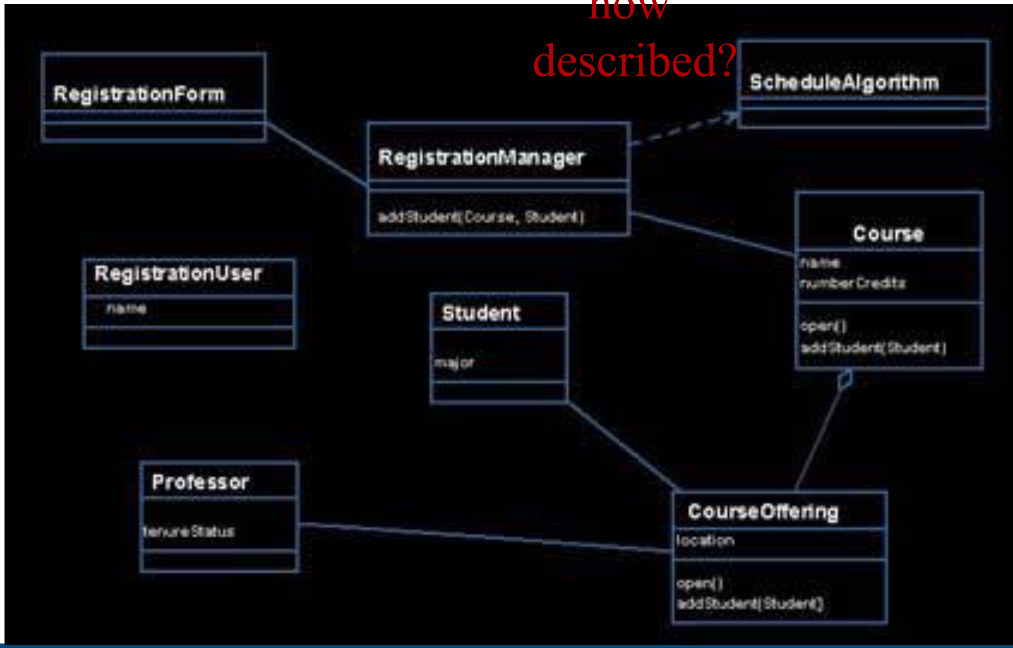
- Class = collection of objects with common structure, common behavior, common relationships, and common semantics
- Displayed as box with up to 3 compartments:
  - Name
  - List of attributes (aka state variables)
  - List of operations
- Class modeling elements include:
  - Classes with structure + behavior
  - Relationships
  - Multiplicity and navigation indicators
  - Role names



Class Diagrams: (Instance) Relships

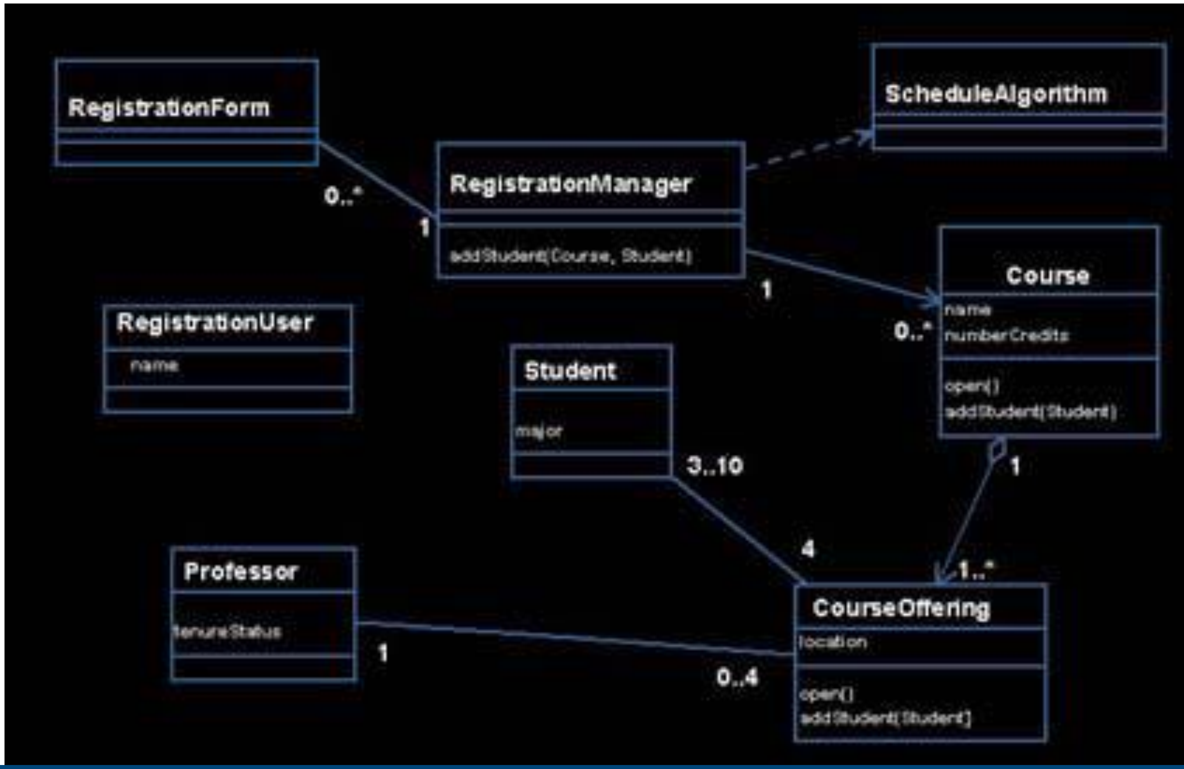
- Models that two objects can “talk”
- Association – bi-directional connection between classes
  - “I can send you a message because if I’m associated with you, I know you’re there.”
- Aggregation – stronger form: „has a“
  - R. between a whole and its parts
- Dependency – weaker form
  - “need your services, but I don’t know that you exist.”
- Quatrani: „*typically first make everything an association, lateron refine*“

how described?



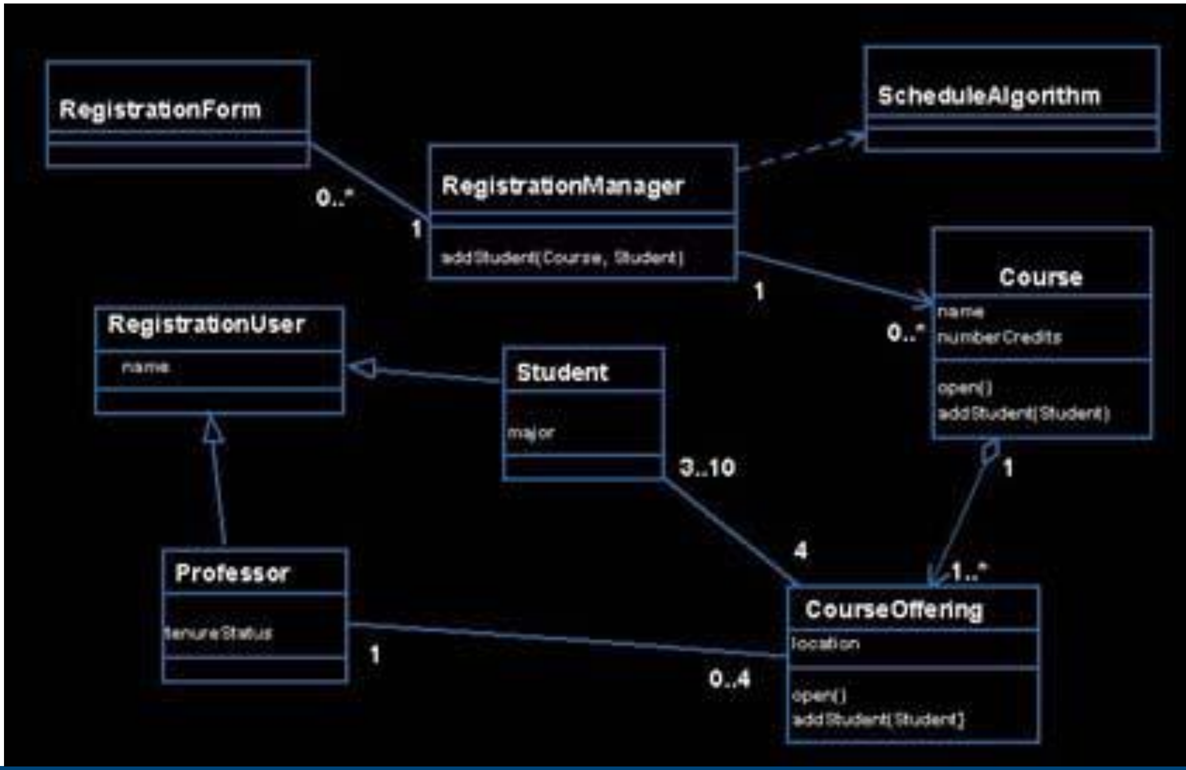
Class Diagrams: Multiplicities, Navig.

- Multiplicity numbers & intervals denote number of instances that can/must participate in relationship instance
  - For both ends of relationship edge
    - *0..1 (may have one)*
    - *1 (must have one)*
    - *0..* or * (may have many)*
    - *1..* (has at least one)*
- Arrow head to denote: traversable only this direction



Class Diagrams: Inheritance

- Inheritance = relation between sub*class* and super*class*
- Subclass instances have
  - all properties specified in superclass
  - plus the specific ones defined with the subclass
- Also called „is-a“



Ready? Let‘s go!

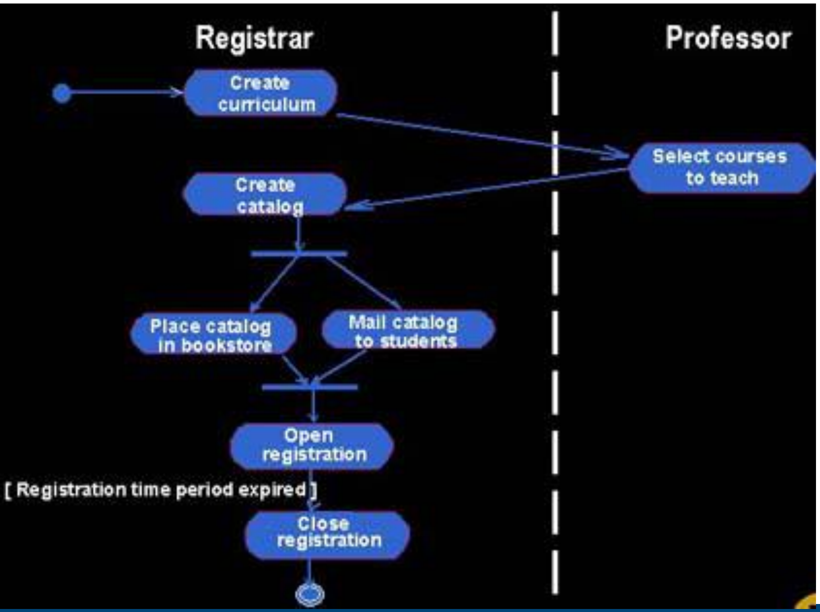
- Trike
- Different types of vehicles
- Family

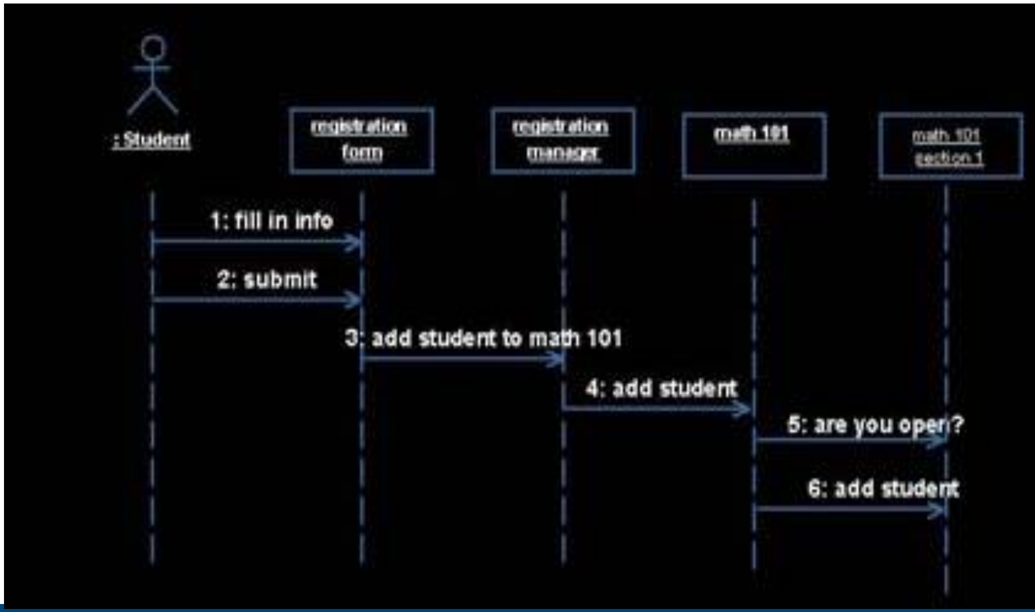
[motorbike-search-engine.co.uk]

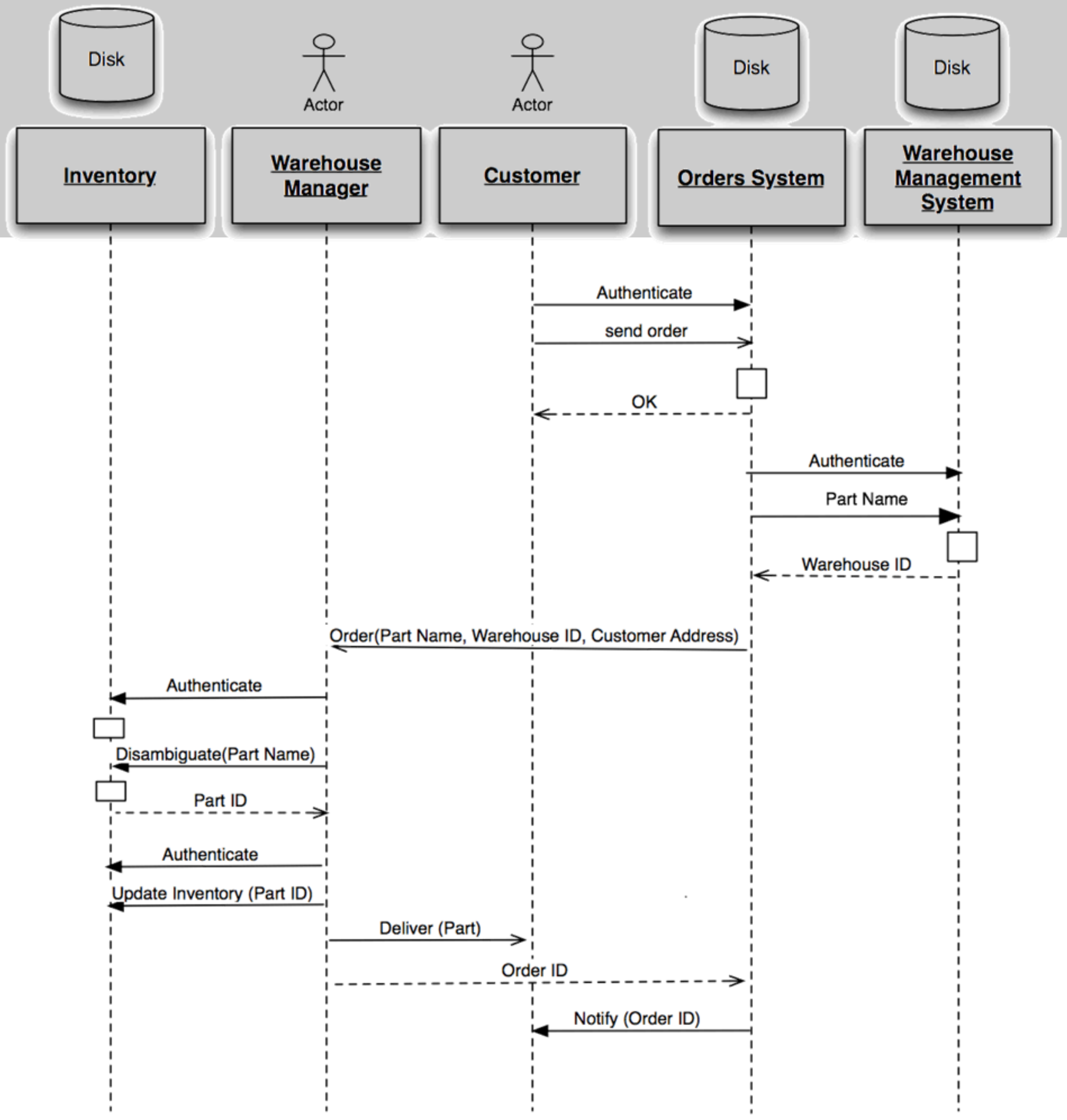
Activity vs Sequence Diagrams?

- Activity diagram:
- Granularity: user-perceived actions
- How do *actors* interact?

- Sequence diagram:
- Granularity: actors + system components
- How do *components* interact?







JACOBS UNIVERSITY

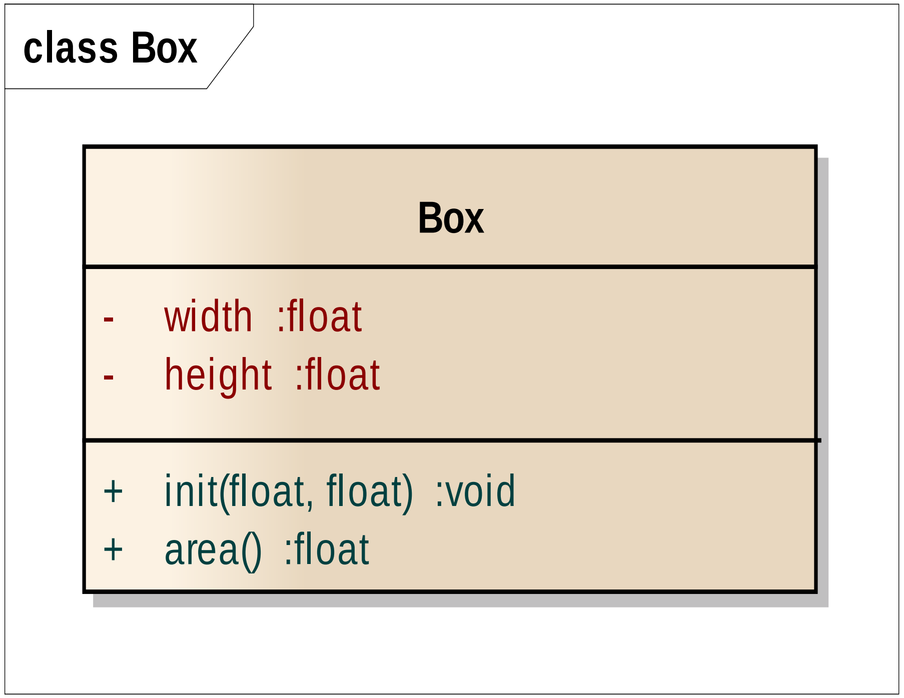
Warehouse Management

[interopporesearch.blogspot.com]

Ready? Let‘s go!

- Customer / waiter / chef

UML → python



```
class Box:
    def __init__(self,w,h)
        self.width = w
        self.height = h
    def area(self)
        return self.width
             * self.height
```

Particularities of python

- Properties are simply used, without declaration
- Inheritance indicated after class name
  - Ex: `class B(A)` means „B in herits from A“
- Private items start with „__“ (2x underscore)
  - Ex: `__myPrivateValue`
- Builtin methods for object maintenance
  - Ex: `__new__()`, `__del__()`, `__repr__()`, *plus many more*
  - Other languages, such as C++, offer default constructors, copy constructors, destructors

Wrap-Up

- UML industry standard
  for visually describing all aspects during software life cycle
  - Use Case Diagram, Activity Diagram , Sequence Diagram, Class Diagram, State Diagram, ...
- We had but a primer – UML spec has ~1,000 pages...
- More work in the beginning (= before coding starts),
  but will pay off in
  - Better design (less flaws & more consistency)
  - Fewer costly surprises late at integration / customer testing time
  - Better plannable
  - Higher customer satisfaction, better career