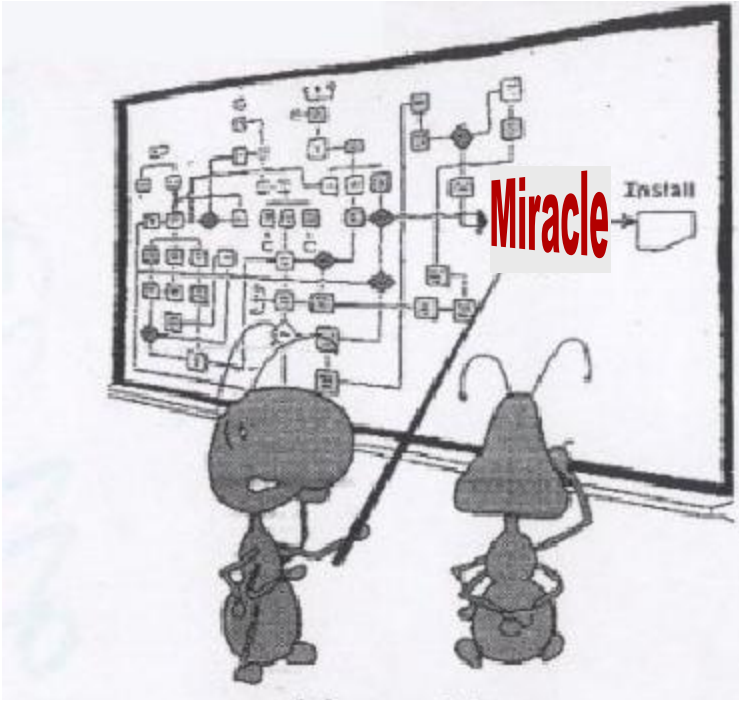
"Plan? Who needs a plan?"
Introduction to UML



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

Instructor: Peter Baumann

email: pbaumann@constructor.university

tel: -3178

office: room 88, Research 1

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

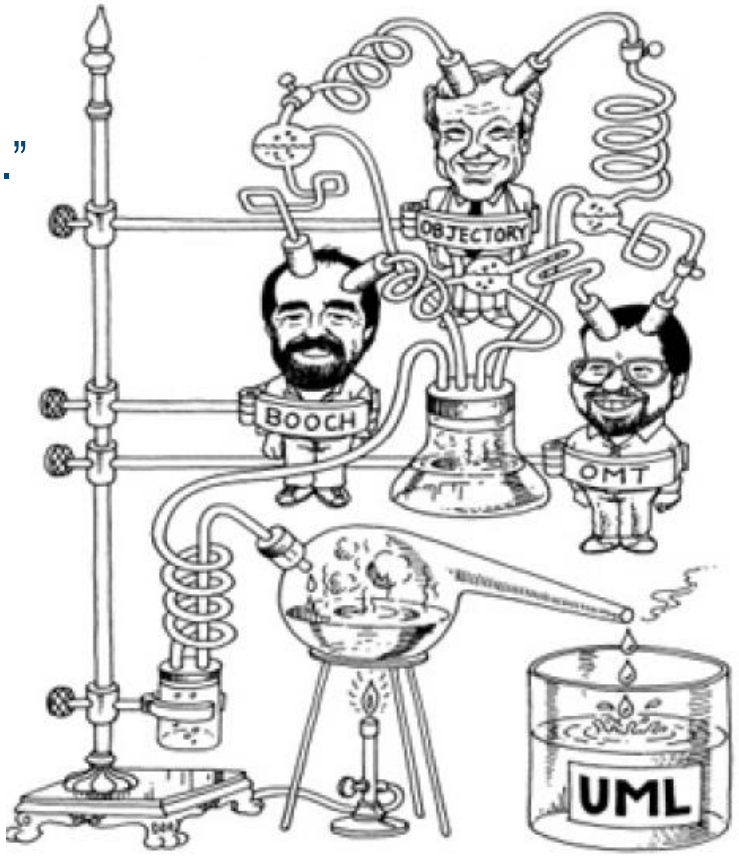
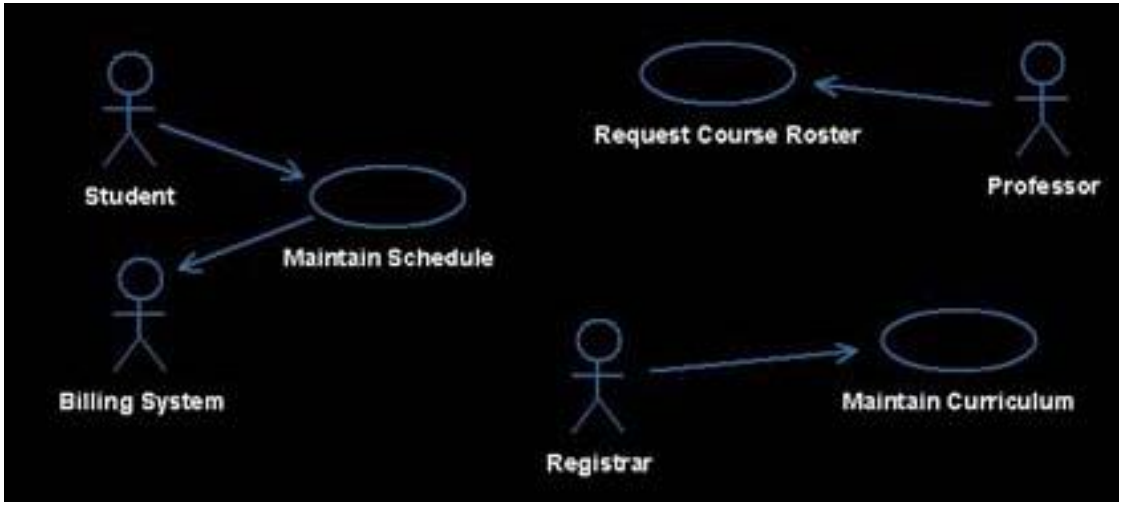


Diagram Types Overview

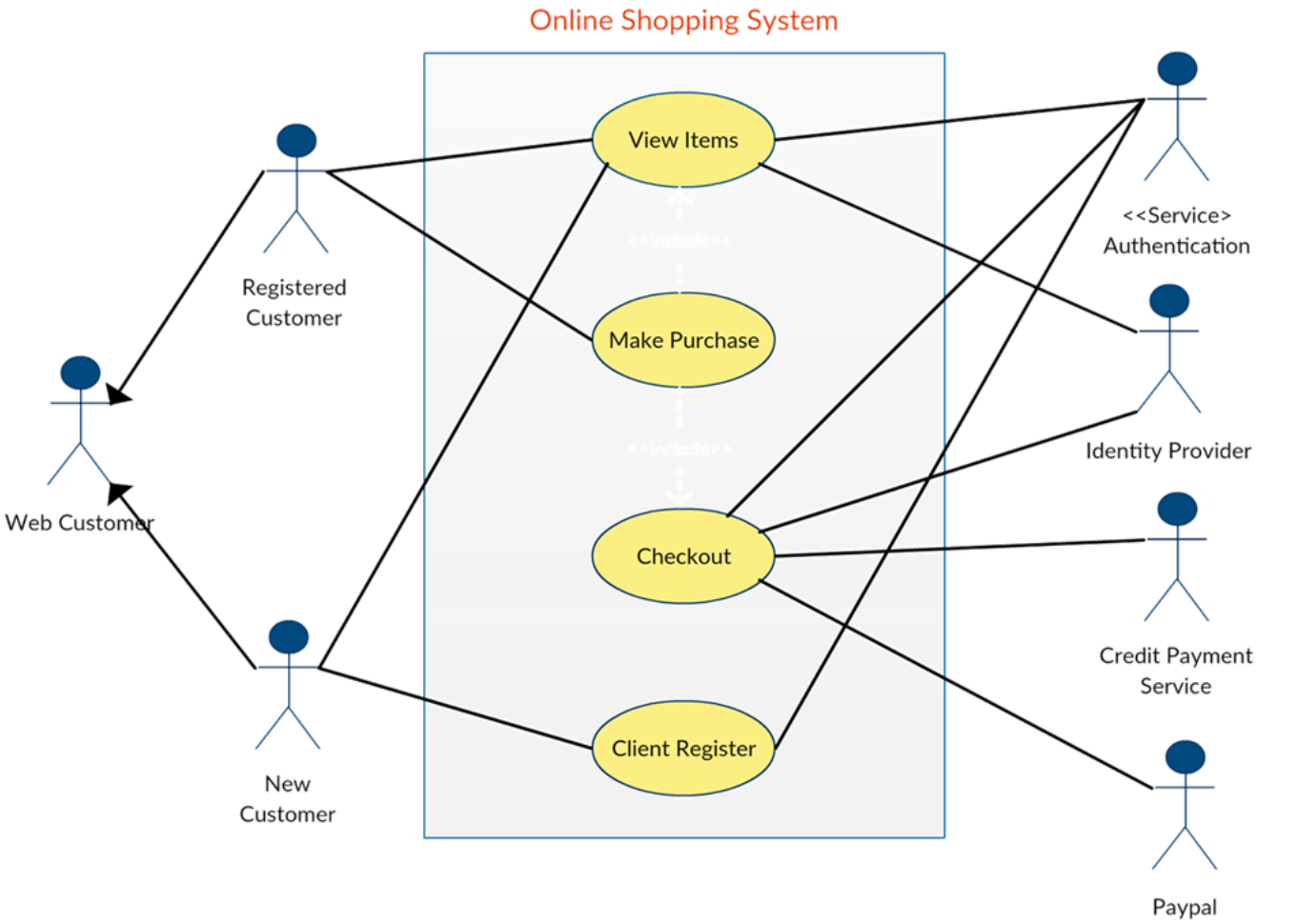
- Main diagram types, according to „80/20 rule“:
  - Use Case Diagram (functional)
  - Activity Diagram (behavioral)
  - Class Diagram (structural)
  - State Diagram (behavioral)
  - Sequence Diagram (behavioral)
- Further, not addressed here:
  - Object Diagram (structural), Collaboration Diagram (structural), 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 !)

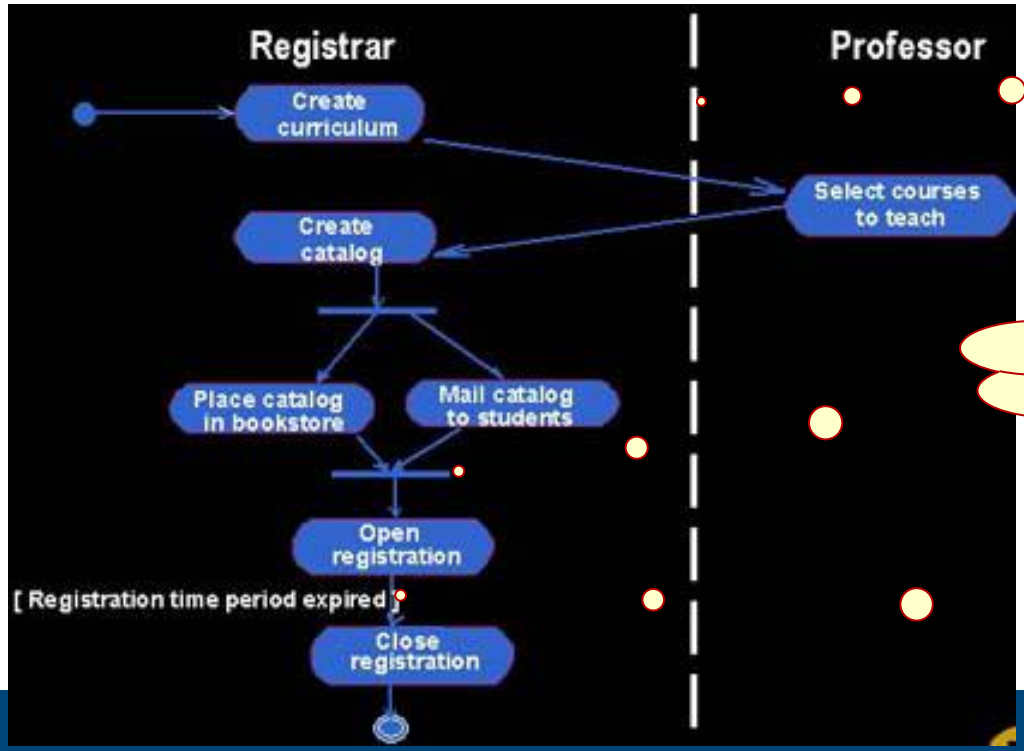


Use Case Diagrams: Larger Example



Activity Diagrams

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



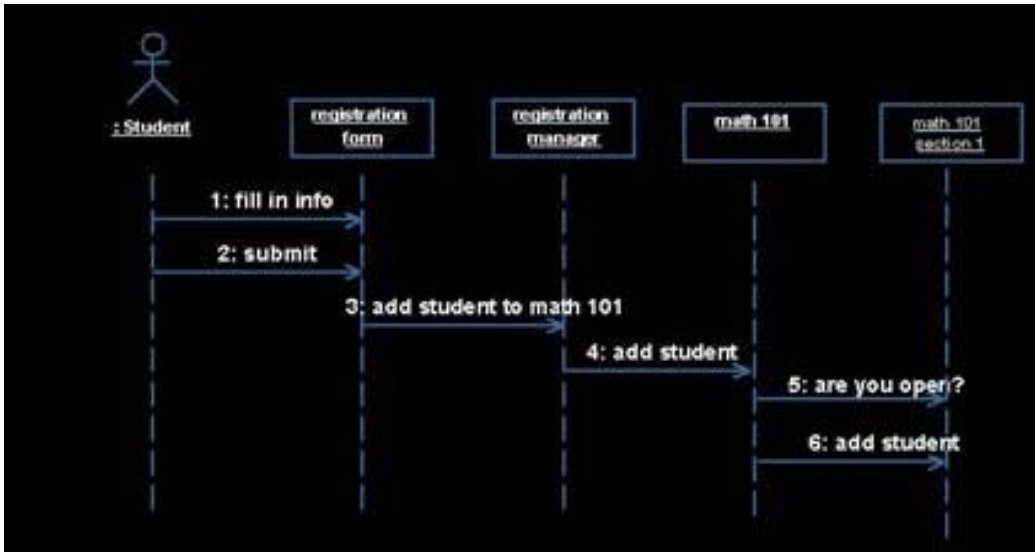
Swimlanes

Synchronisation bar (fork/join)

Transition guard

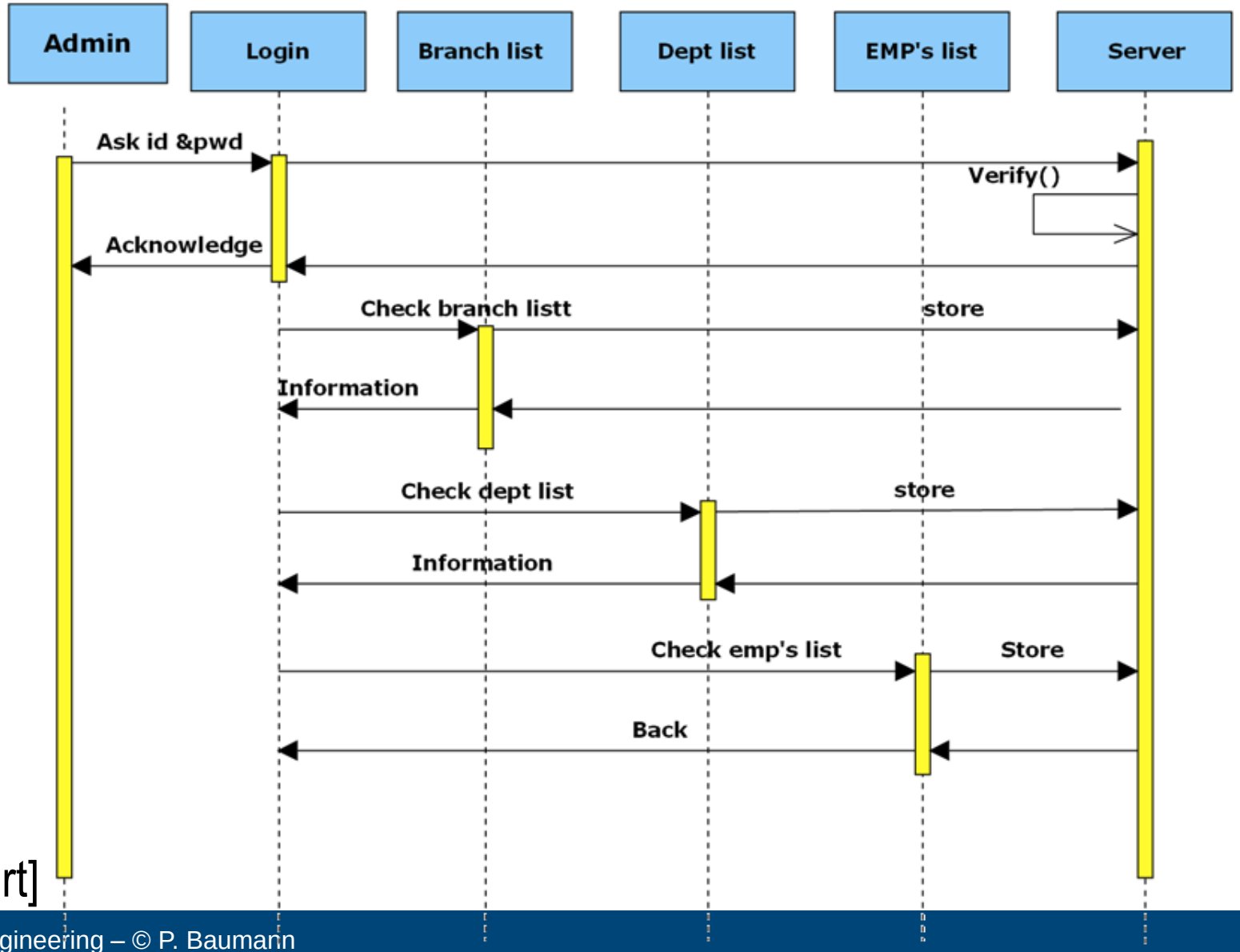
Sequence Diagrams

- 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?*"

Sequence Diagrams: Larger Example



[lucidchart]

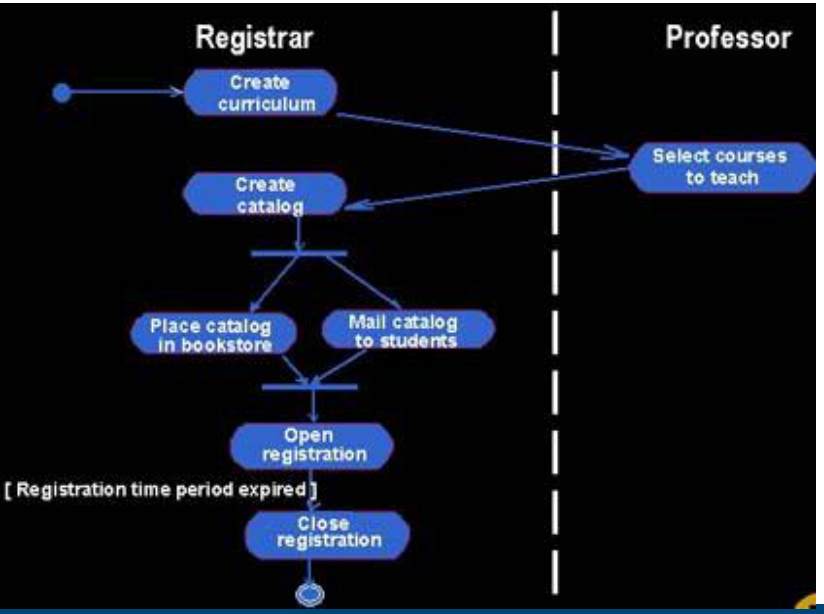
Activity vs Sequence Diagrams?

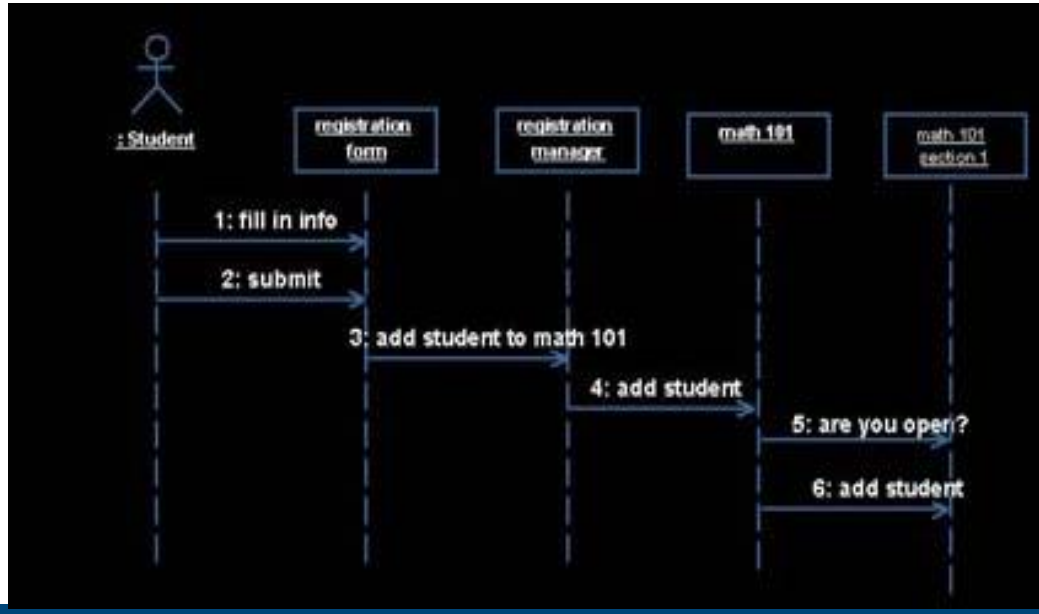
Activity diagram:

- Granularity: user-perceived actions
- Emphasis on internal state transitions

Sequence diagram:

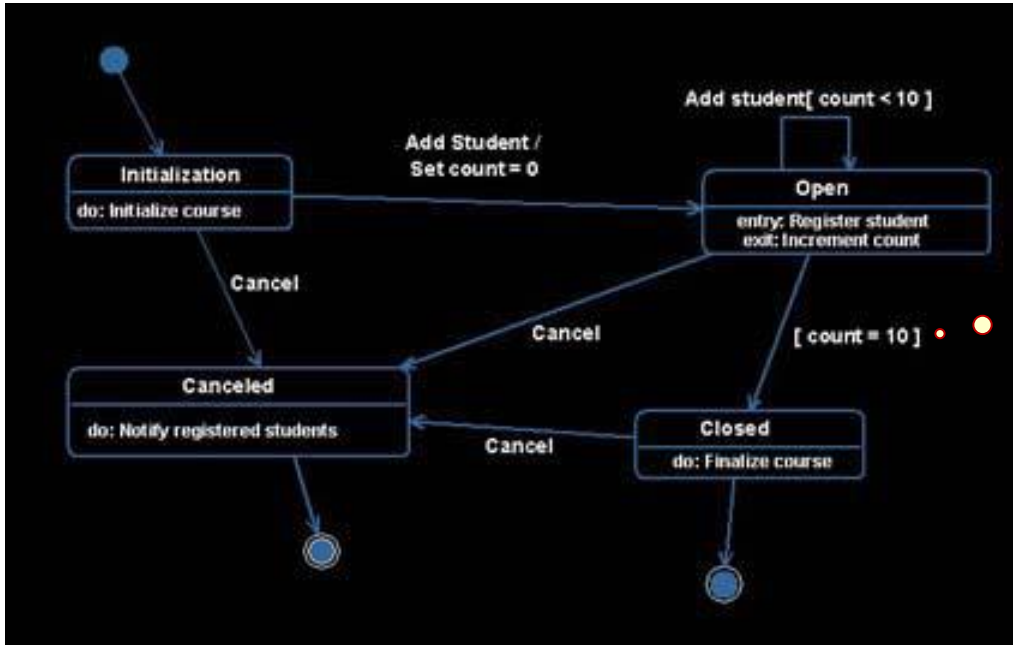
- Granularity: actors + system components
- Emphasis on component interaction





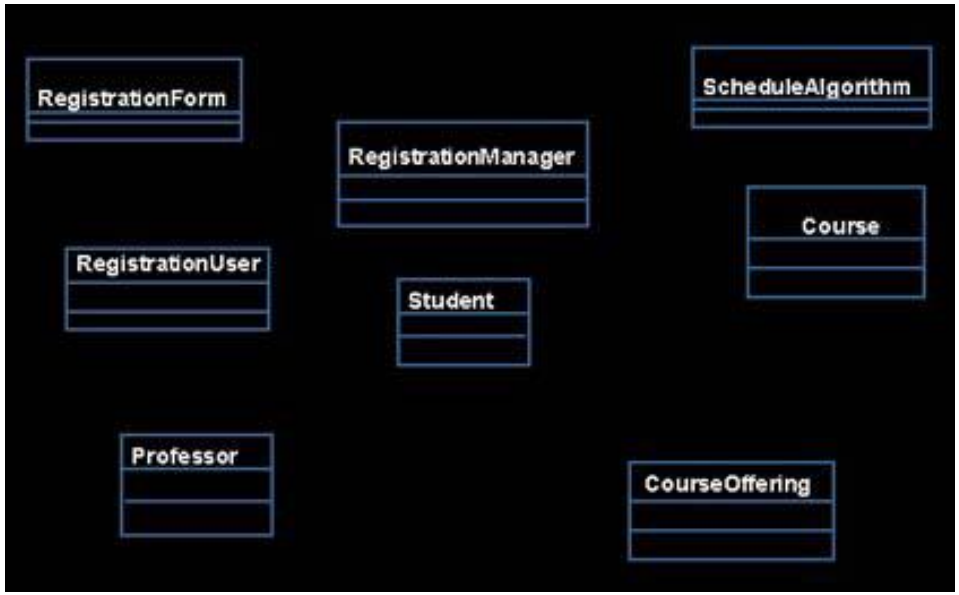
State Transition Diagrams

- show life history of a given class
- use for classes that typically have a lot of dynamic behavior
  - Sequence Diagram: class that's on a lot of sequence diagrams, getting and sending a lot of messages is candidate



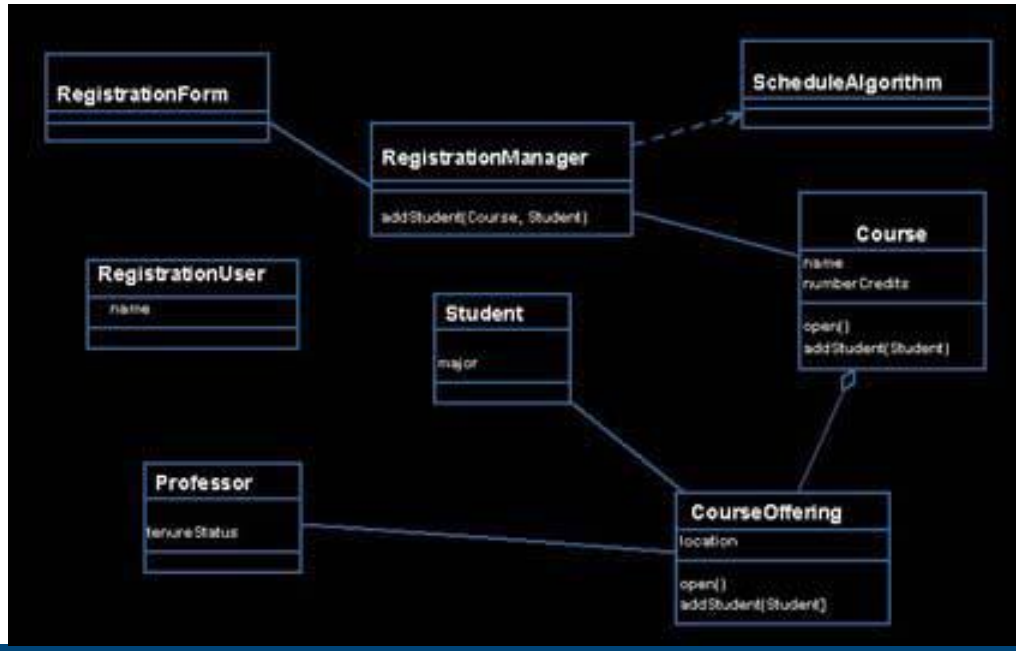
Class Diagrams

- 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



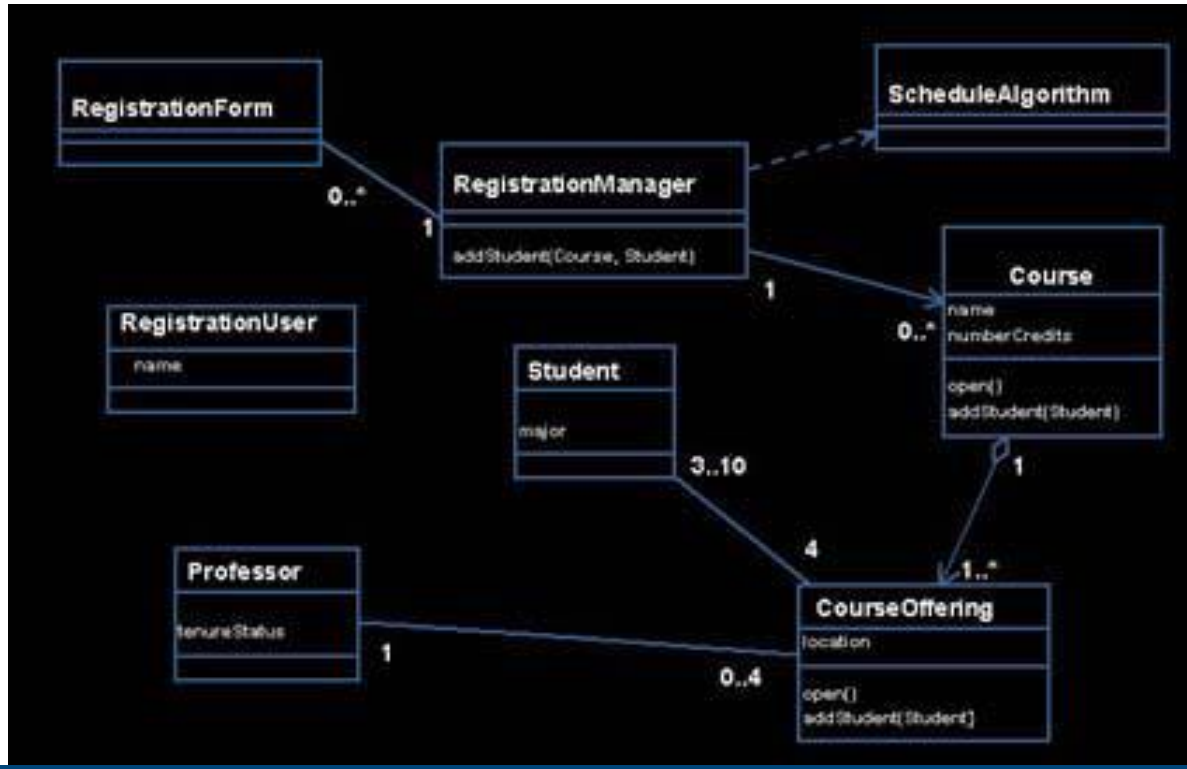
Class Diagrams: (Instance) Relships

- 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*“



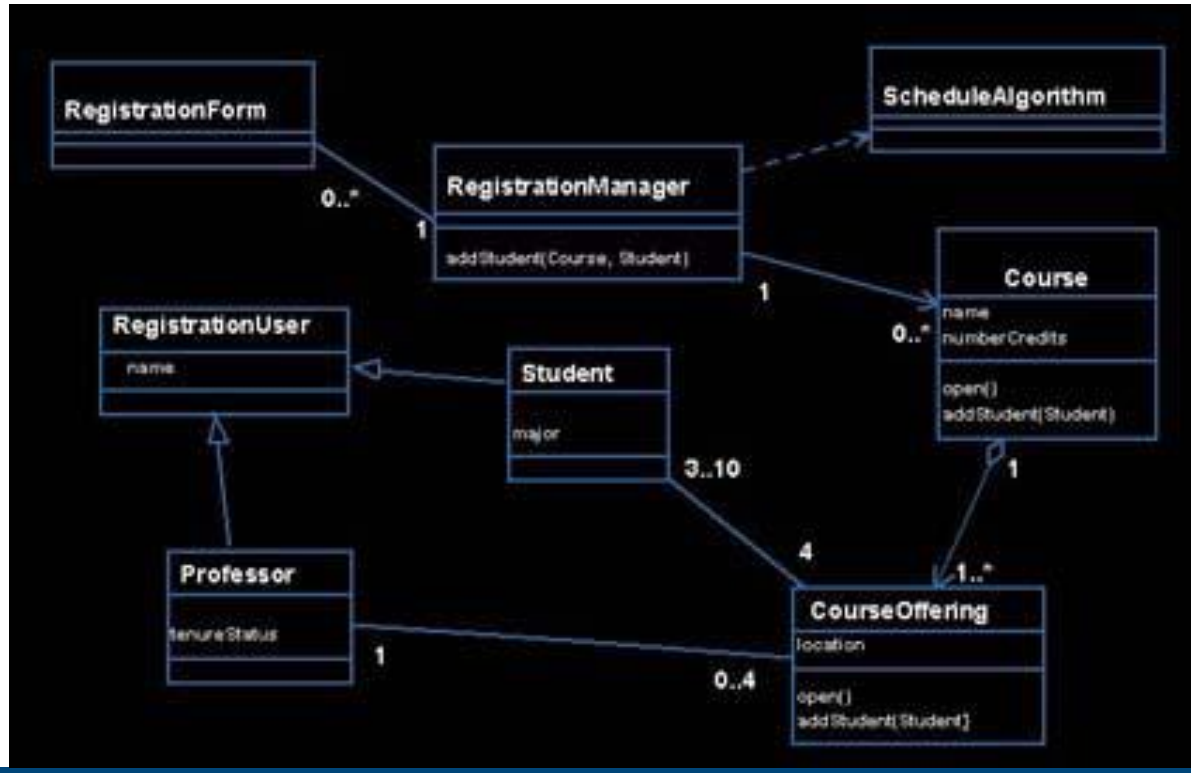
Class Diagrams: Multiplicities, Navigation

- 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

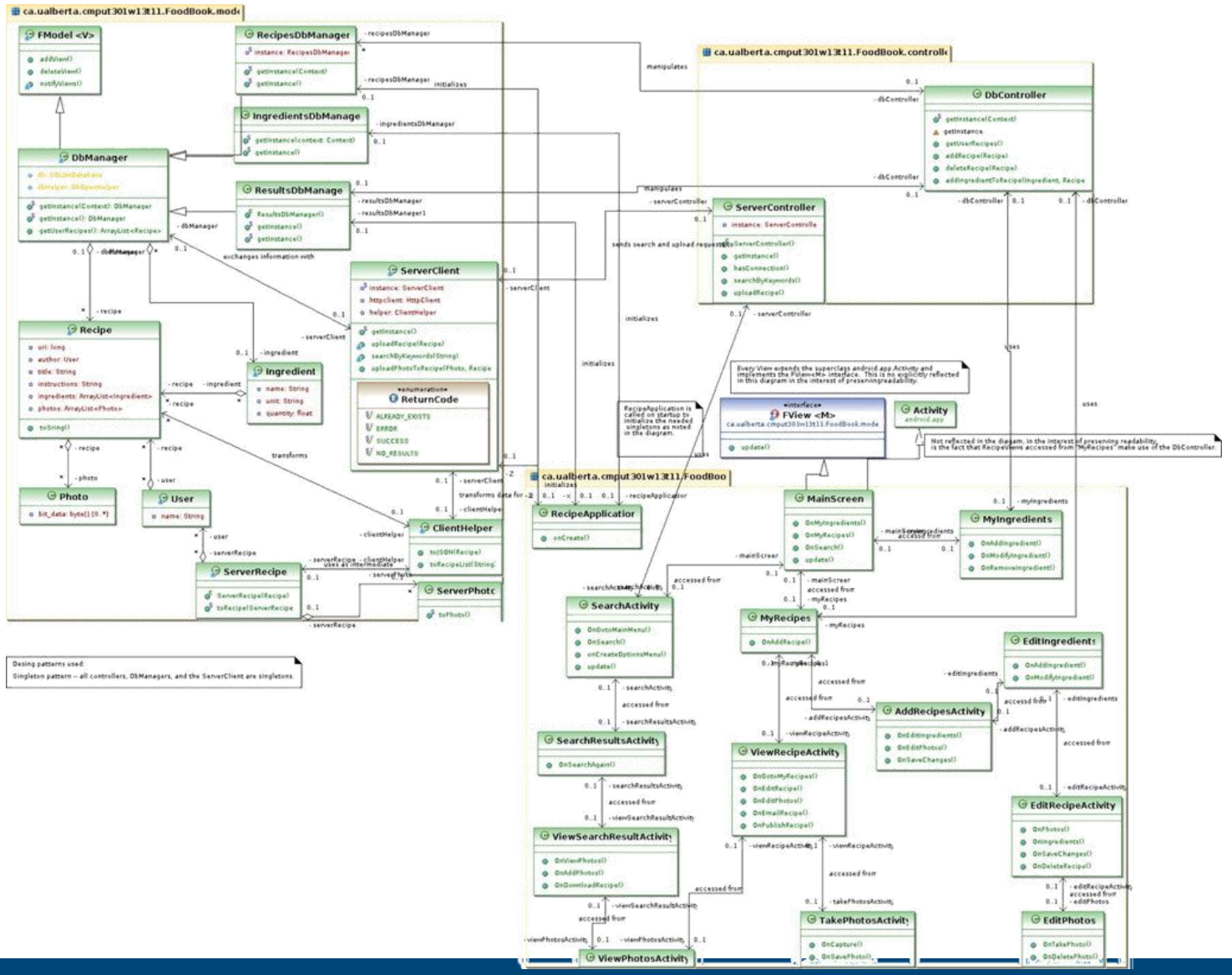


Class Diagrams: Inheritance

- 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“



Class Diagrams: Larger Example



Re-Iterating...

- UML = several diagram types to capture different aspects of sw system
  - Structural, functional, behavioral
- Mutual interrelations
  - use them to do consistency & plausibility cross checking!
- *Fine so far? Let‘s go on...*

Outlook: xUML

- (subset of) UML + executable semantics + timing rules
- Approach: software development method + abstract language
- Advantages:
  - High-level description serves as documentation
  - Translation: platform-independent models (PIM) →platform-specific models (PSM)
- Note: Generalizations always notated as {complete, disjoint} ← DBWS

Outlook: DSLs

SQL is a DSL, optimized for (tabular) data

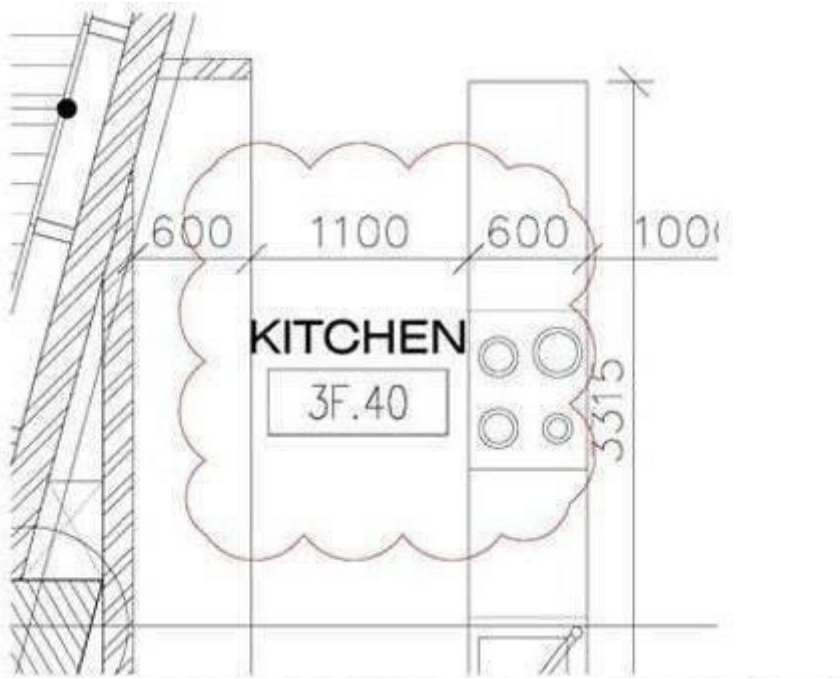
- Alternative to UML for describing systems : domain-specific modelling languages (DSLs)
  - UML considered (too) complex (general-purpose), software biased
- Ex: SysML = general-purpose modelling language for systems engineering applications [sysml.org]
  - UML dialect for hardware, information, processes, personnel, facilities
  - Ex: aerospace, defense, automotive, ...
- Rule of thumb:
  - UML better for enterprise apps (millions of possible directions)
  - DSLs better for embedded systems (focused app domain & paths)

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, ...
- 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

Caveat: Symbology Interpretation

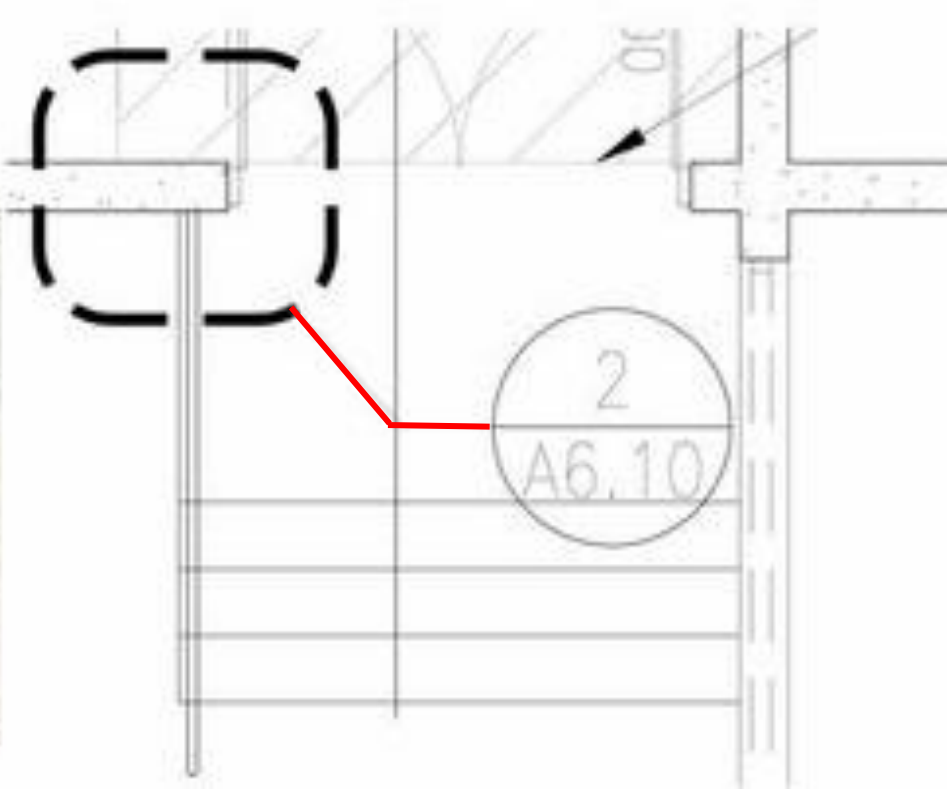
- „revision cloud“ common in mechanical engineering



[autodesk.blogs.com]

Caveat: Symbology Interpretation

- „revision cloud“ common in mechanical engineering



[autodesk.blogs.com]