

"Plan? Who needs a plan?"

Introduction to UML

based on:

Introduction to the Unified Modeling Language, Chapter 2

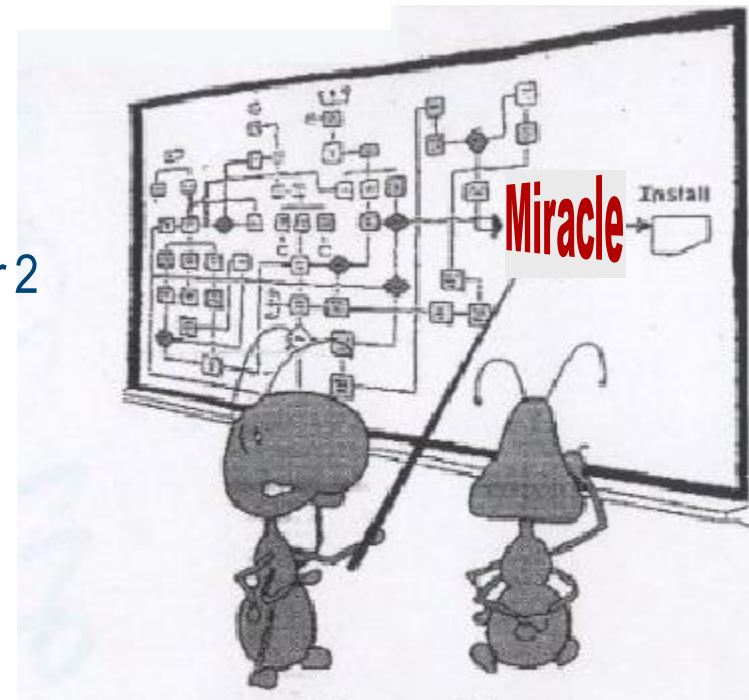
Terry Quatrani, UML Evangelist, IBM

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

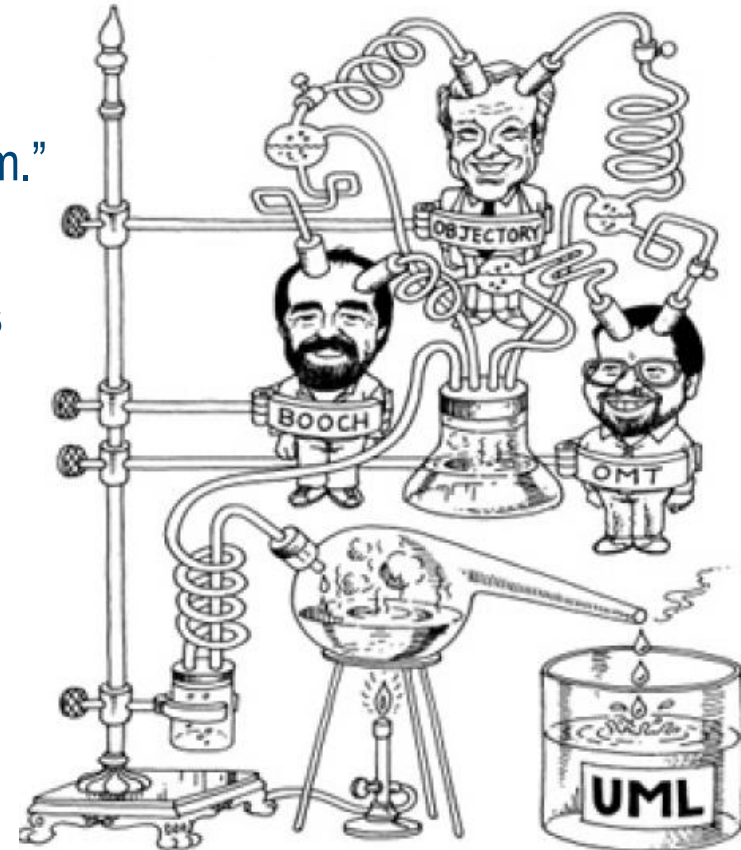
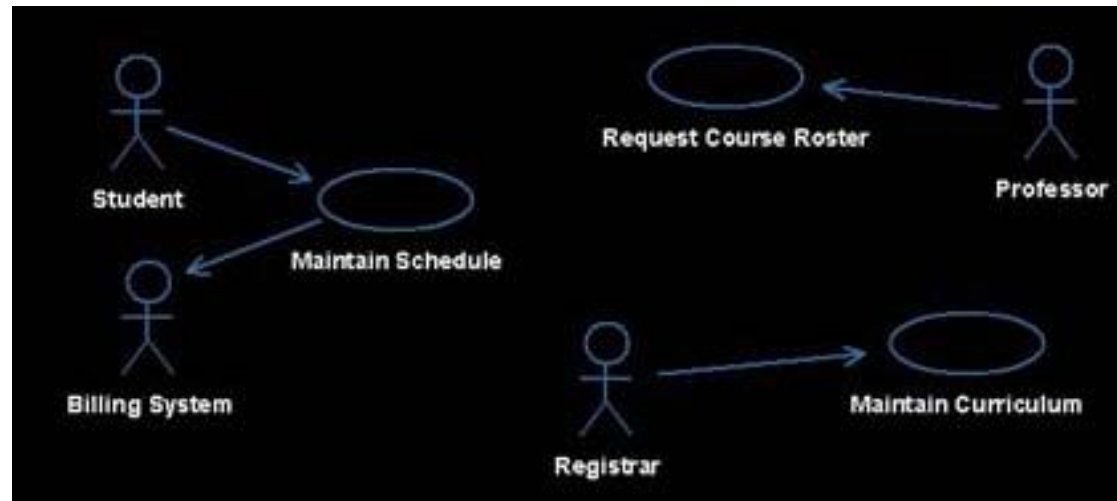


Diagram Types Overview

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

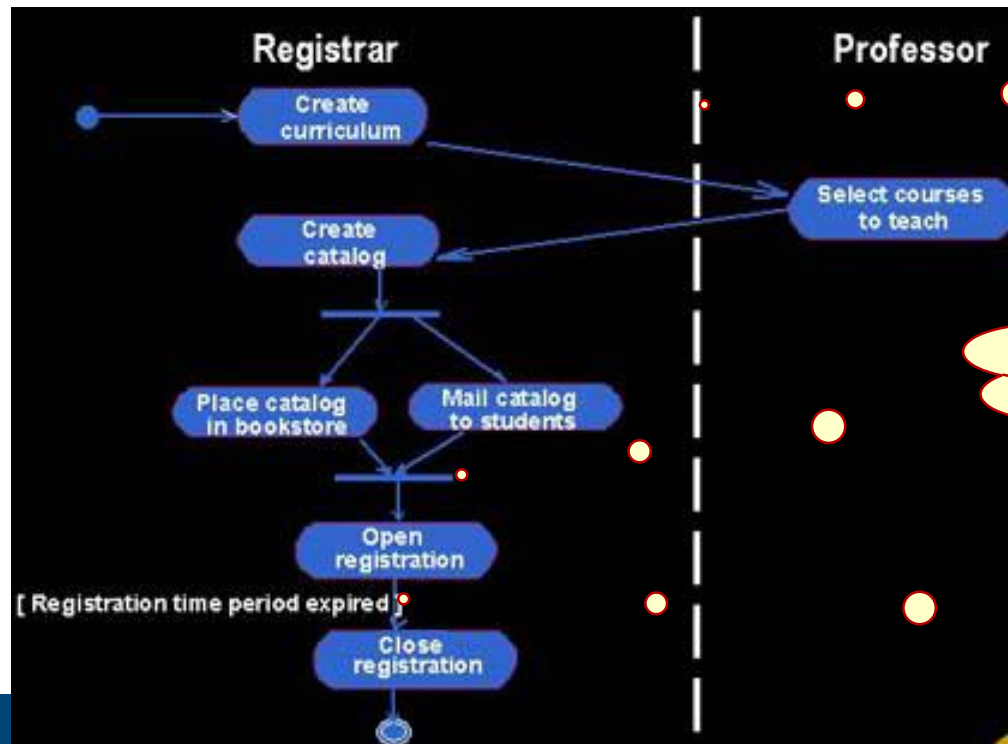
Use Case Diagrams

- **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 !)



Activity Diagrams

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



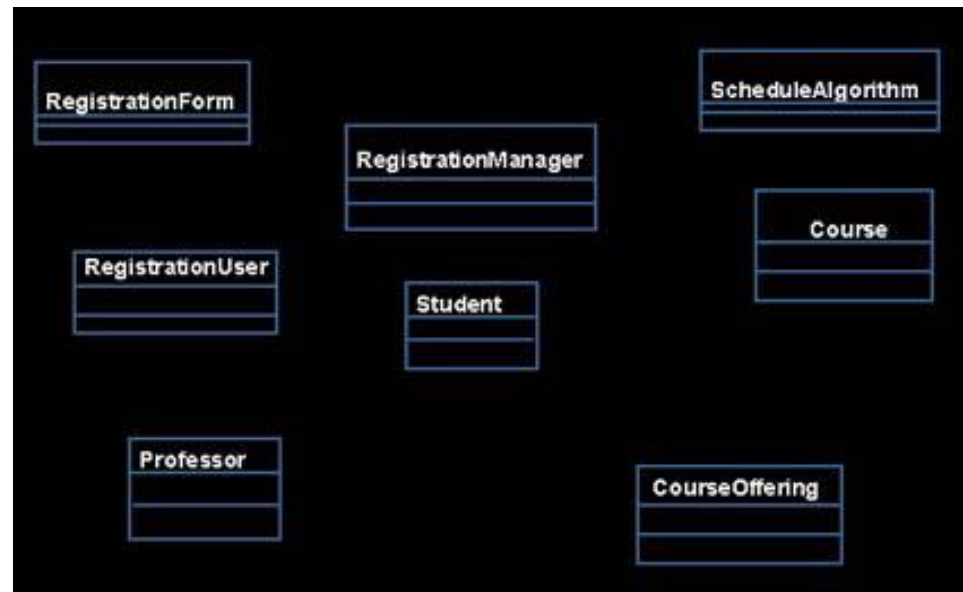
Swimlanes

Synchronisation bar
(fork/join)

Transition
guard

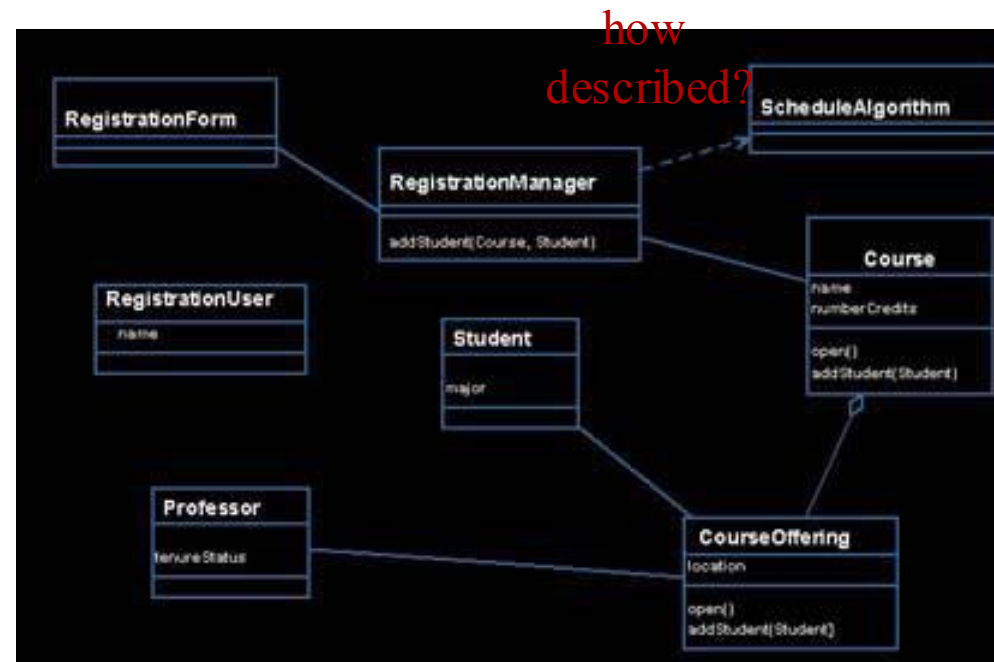
Class Diagrams

- **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



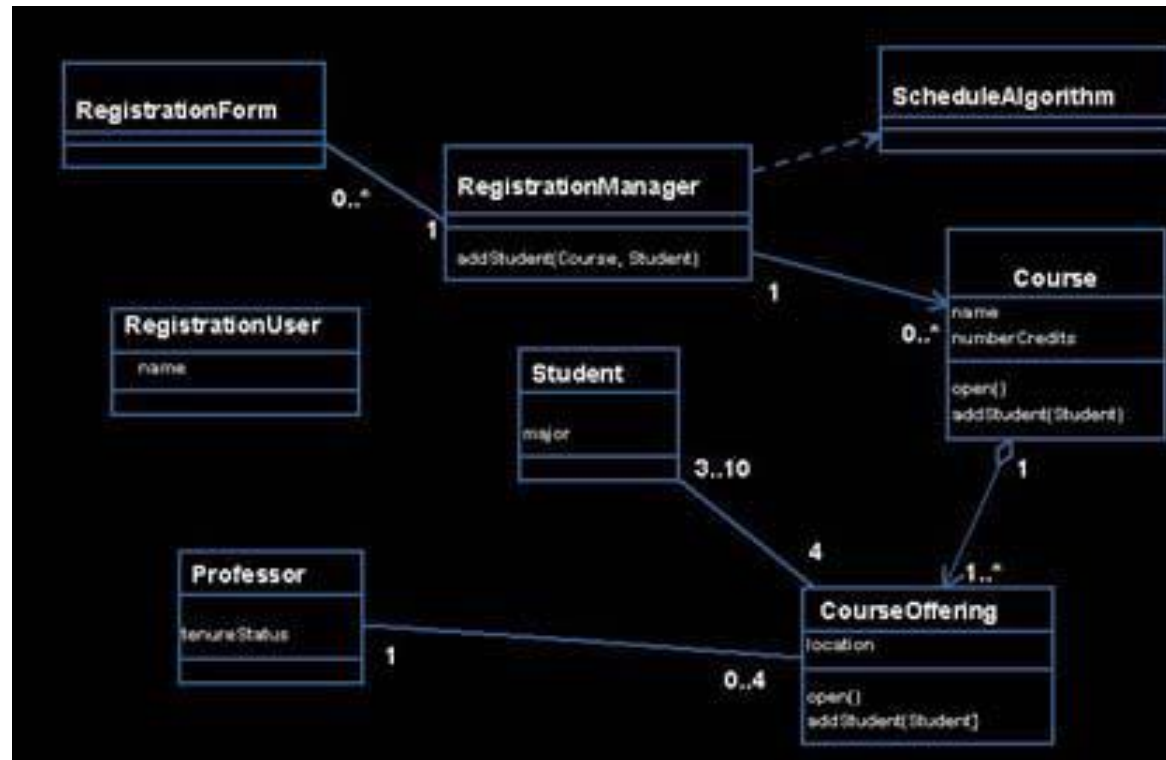
Class Diagrams: (Instance) Relationships

- 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, later on refine“



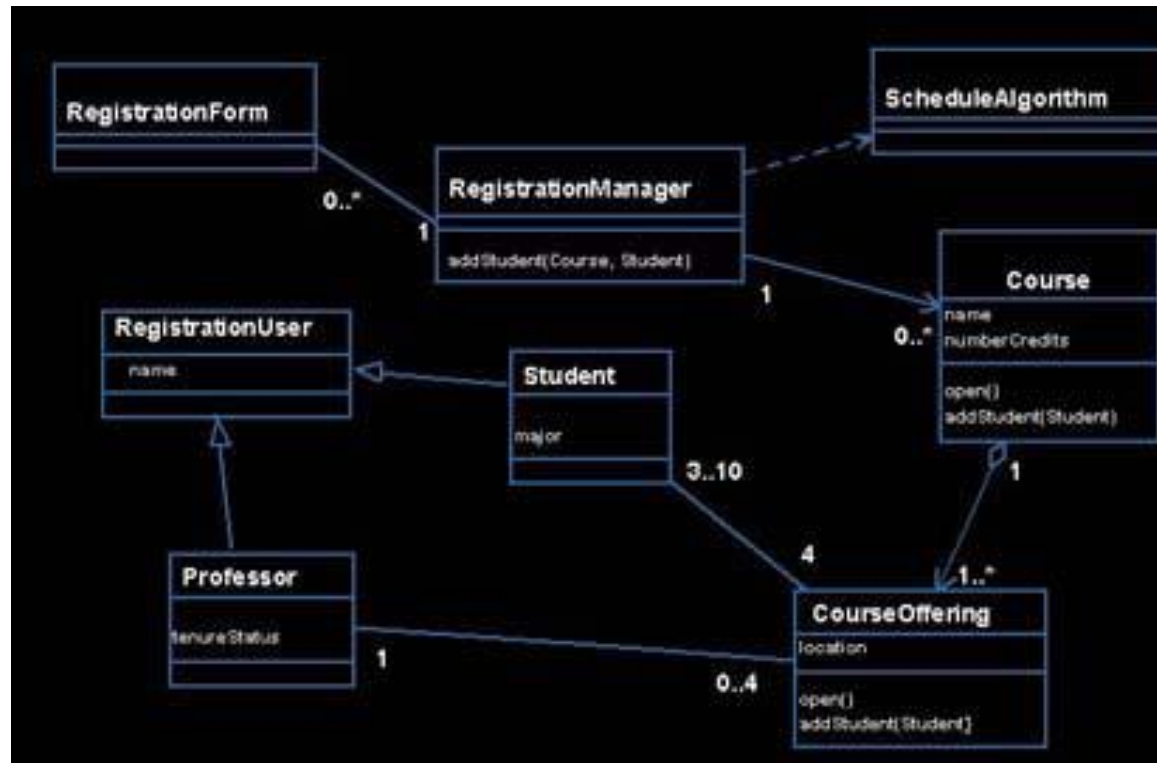
Class Diagrams: Multiplicities, Navig.

- **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 subclass and superclass
- Subclass instances have
 - all properties specified in superclass
 - plus the specific ones defined with the subclass
- Also called „is-a“



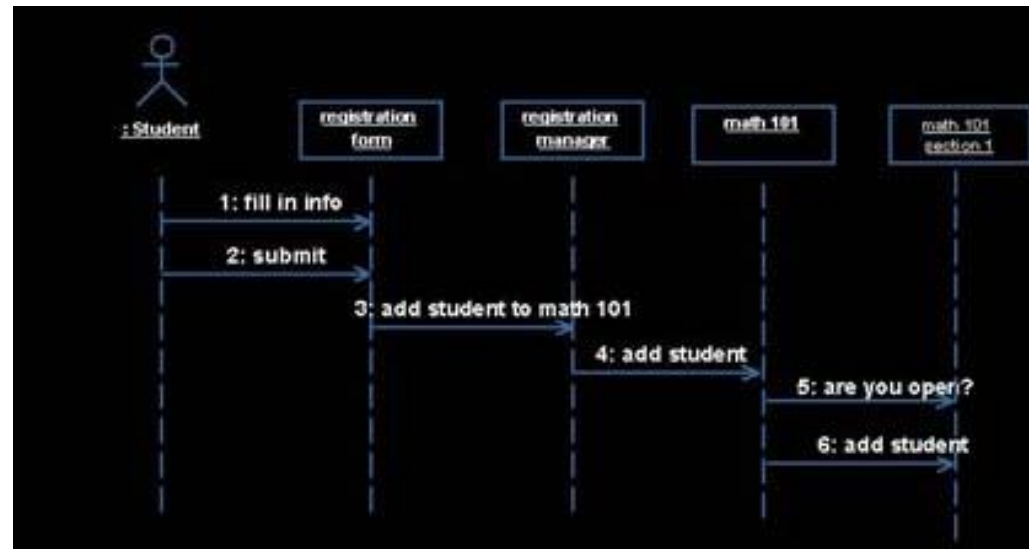
Sequence Diagrams

- 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

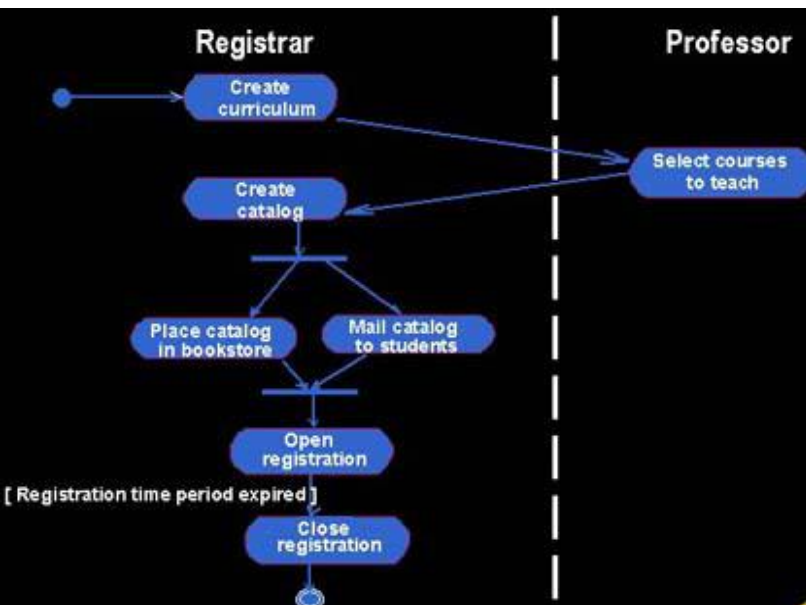


- Useful for designer and customer to answer the question:
„what objects and interactions will I need to accomplish the functionality specified by the flow of events?“

Activity vs Sequence Diagrams?

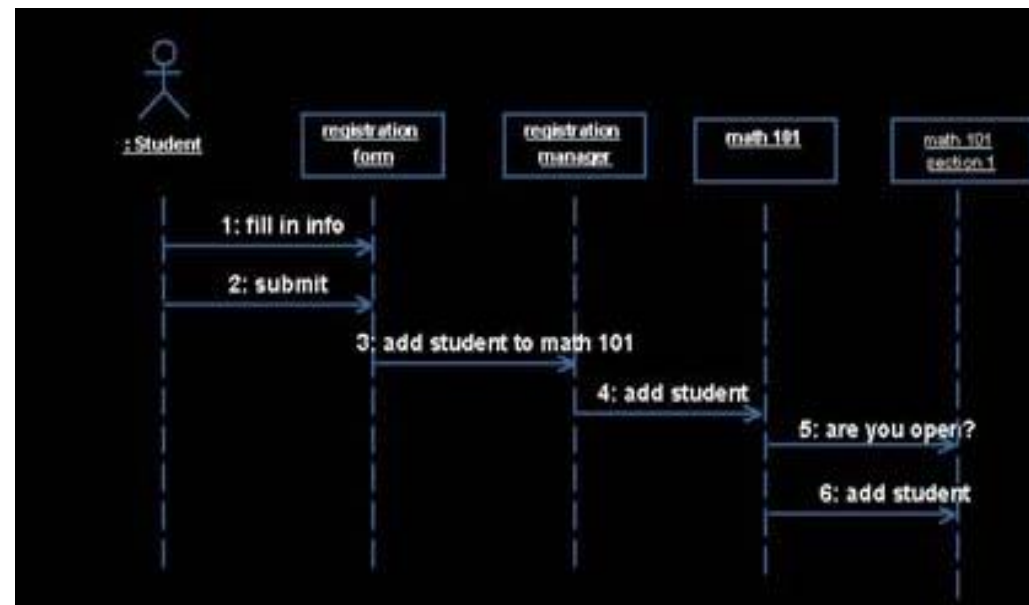
Activity diagram:

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



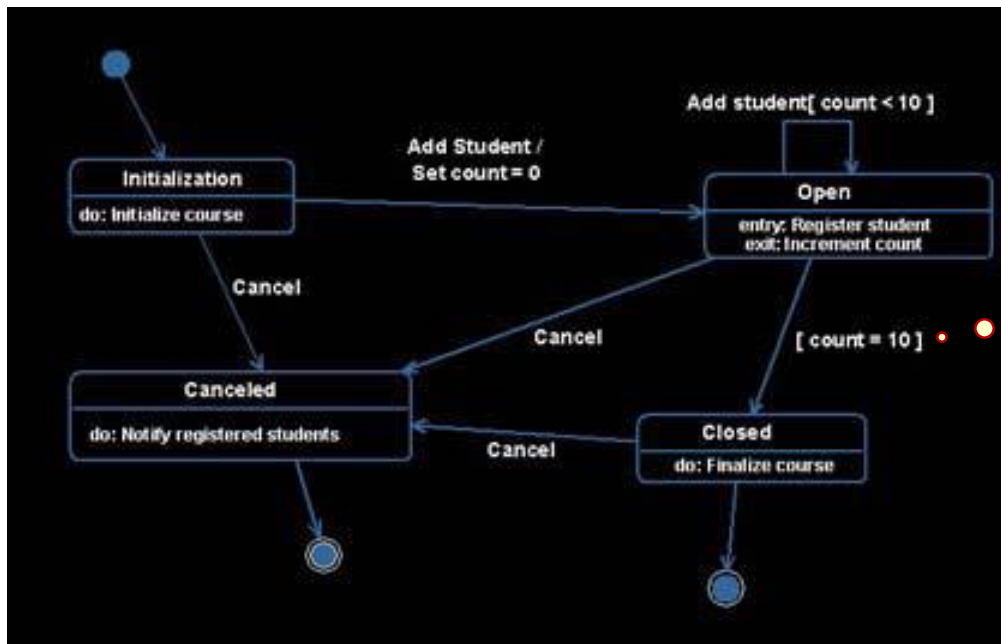
Sequence diagram:

- 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



guard

Re-Iterating...

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

Outlook: UML 2.0

- Substantially revised, in particular for **Model-Driven Architecture**
 - **Infrastructure**: core of architecture, profiles, stereotypes
 - **Superstructure**: static & dynamic model elements
 - **Object Constraint Language (OCL)**: formalize assertions, rules
 - **Diagram Interchange**: UML exchange format
- Vision:
UML spec → platform-indepd. model → target model → implementation

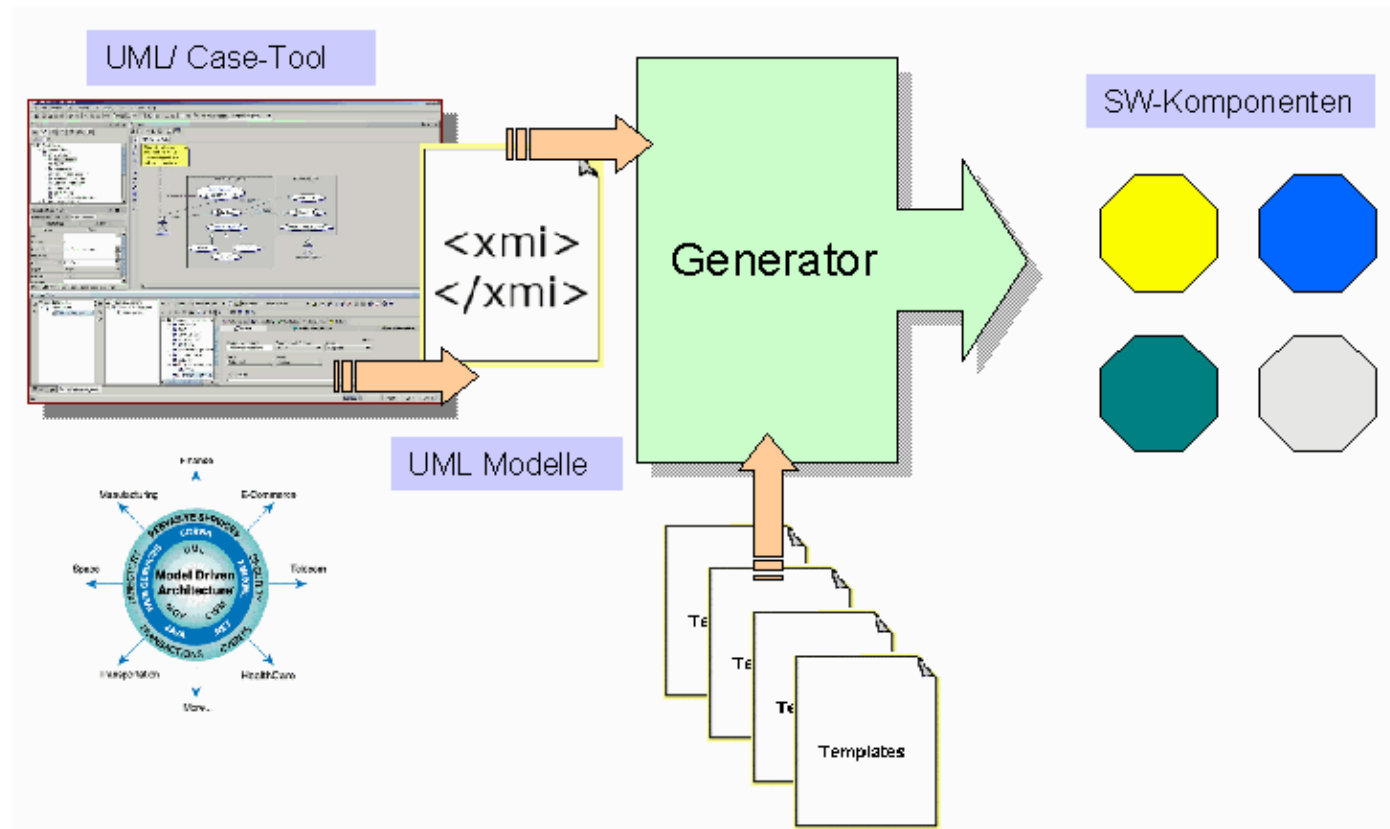
PIM

PSM

 - verify & validate specs already in design phase
- Main goals:
speed up process, higher quality, reusability, long-term usability

Outlook: UML 2.0 (contd.)

- The „ideal UML/MDA machine“:



Outlook: DSLs

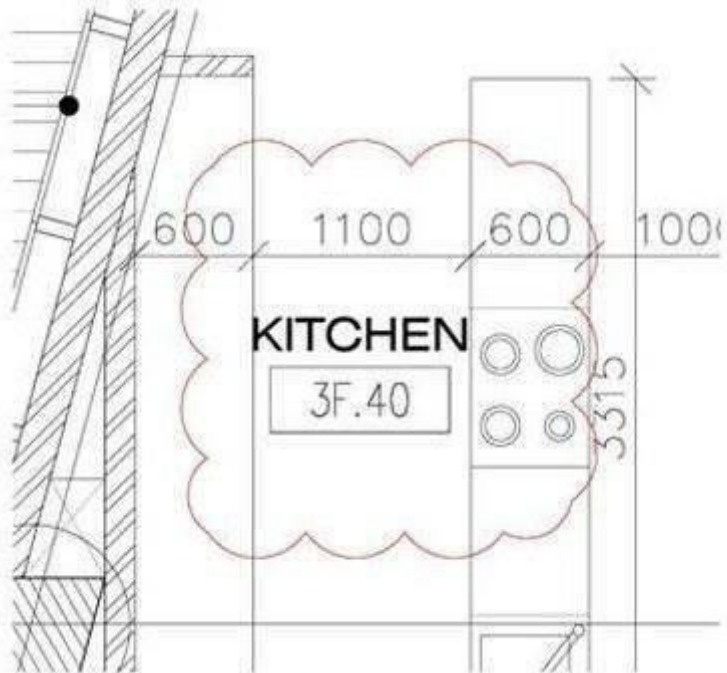
- Alternative to UML for describing systems :
domain-specific modelling languages (DSLs)
 - UML considered (too) complex (general-purpose), software biased
- Ex. **SysML** = general purpose modelling language for **systems engineering** applications [www.sysml.org]
 - SysML emphasizes hardware, information, processes, personnel, facilities
 - UML dialect, issued by OMG
 - Used, eg, in aerospace, defense, automotive
- Rule of thumb:
 - UML better for enterprise apps (millions of possible directions)
 - DSLs better for embedded systems (clearly delimited 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

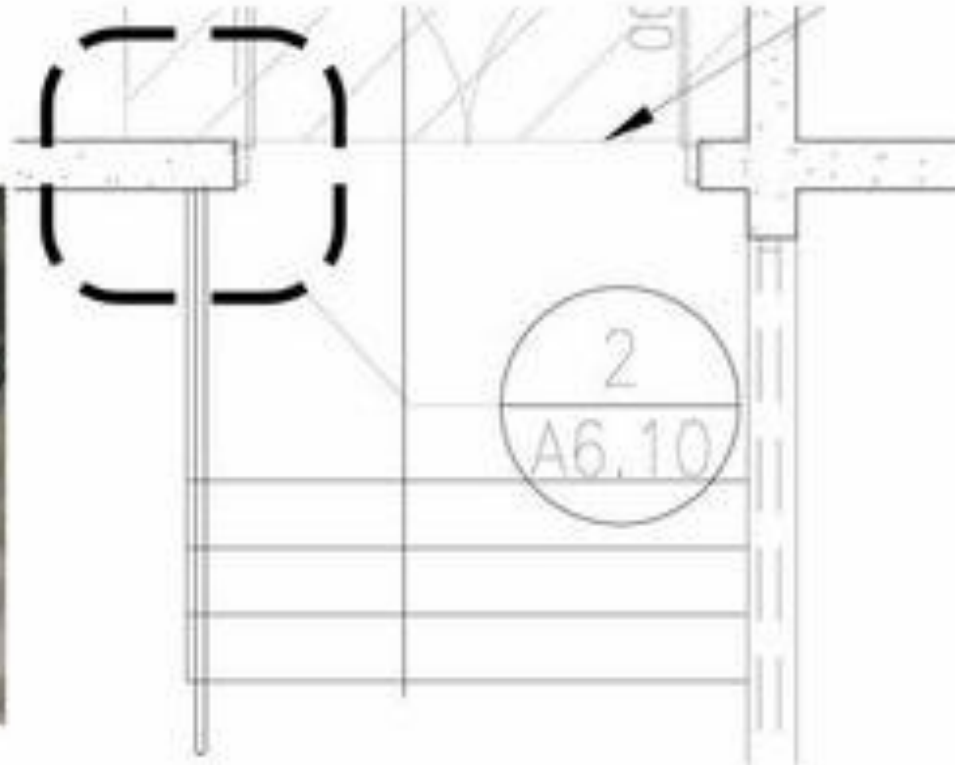
Caveat: Symbology Interpretation

- „revision cloud“ common in mechanical engineering



Caveat: Symbology Interpretation

- „revision cloud“ common in mechanical engineering



[autodesk.blogs.com]

Further Reading

- We had but a primer – UML spec has ~700 pages...
- See my course web page for a list of tutorials etc.
 - Introduction to the Unified Modeling Language, by Terry Quatrani
 - *article from where this presentation was made*
- Books:
 - Visual Modeling with Rational Rose 2002 and UML, by Terry Quatrani
 - UML Distilled, by Martin Fowler
 - UML Explained, by Kendall Scott
- www.uml.org