Politecnico
di Milano

Elicitation of requirements

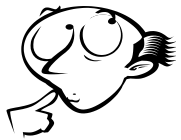
Role of requirements models

stakeholders

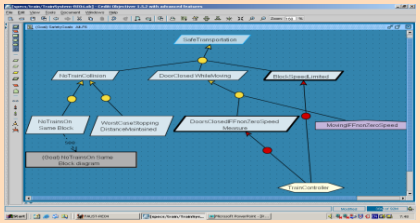
existing systems

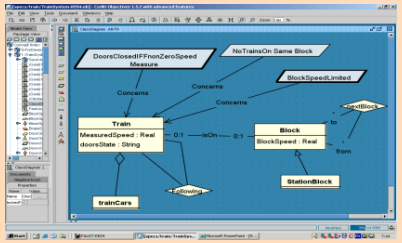
documents

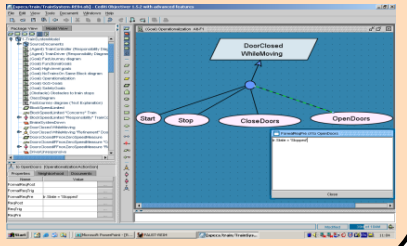
elicitation
& modelling



Requirements Models

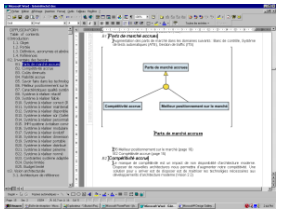






analysis
& validation

generation of
RE deliverables



requirements document

Complexity example: rules and procedures for loan approval

- Implicit knowledge:
  - *There is no document in which the rules for approving loans are written down or the documents are ambiguous, incomplete, outdated*
- Conflicting information:
  - *Different members of the department have different ideas about what the rules are*
- Bias:
  - *The loan approval officers fear that your job is to computerize their jobs out of existence, so they are deliberately emphasizing the need for case-by-case discretion (to convince you it has to be done by a human!)*
- Tacit knowledge:
  - *The loan approval process described to you by the loan approval officers is quite different from your observations of what they actually do*
- Probe effect:
  - *The loan approval process used by the officers while you are observing is different from the one they normally use*

Scenarios

- “A narrative description of what people do and experience as they try to make use of computer systems and applications” [M. Carrol, Scenario-based Design, Wiley, 1995]

- A concrete, focused, informal description of a single feature of the system

Scenario example: warehouse on fire

- *Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car.*

- *Alice enters the address of the building, a brief description of its location (i.e., north west corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appears to be relatively busy. She confirms her input and waits for an acknowledgment.*

- *John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated time of arrival (ETA) to Alice.*

- *Alice received the acknowledgment and the ETA.*

Observations

- Concrete scenario
  - Describes a single instance of reporting a fire incident
  - Does not describe all possible situations in which a fire can be reported
- Participating actors
  - Bob, Alice, and John

Questions for identifying actors

- Which user groups are supported by the system to perform their work?
- Which user groups execute the system's main functions?
- Which user groups perform secondary functions such as maintenance and administration?
- With what external hardware or software system will the system-to-be interact?

Heuristics for finding scenarios

- Ask yourself or the client the following questions:
  - What are the primary tasks that the system needs to perform?
  - What data will the actor create, store, change, remove or add in the system?
  - What external changes does the system need to know about?
  - What changes or events will the actor of the system need to be informed about?
- However, don't rely on questionnaires alone
- Insist on **task observation** if the system already exists (interface engineering or reengineering)
  - Ask to speak to the end user, not just to the software contractor
  - Expect resistance and try to overcome it

So far…

- Scenarios provide a nice summary of what the requirements analysis team can derive from
  - Observation
  - Interviews
  - Analysis of documentation
- … they can be very specific
- How to abstract from details and specificities?
  - … Use cases!

After the scenarios are formulated…

- Generalize scenarios
  - Example: from “Warehouse on fire” we can generalize a “Report Emergency” use case
- Describe each of these use cases in more detail
  - Participating actors
  - Describe the Entry Condition
  - Describe the Flow of Events
  - Describe the Exit Condition
  - Describe Exceptions
  - Describe Special Requirements (Constraints, Nonfunctional Requirements)

Use cases

- A use case is a flow of events in the system, including interaction with actors
- It is initiated by an actor
- Each use case has a name
- Each use case has a termination condition

Use Case Model: The set of all use cases specifying the complete functionality of the system

Use case example: ReportEmergency

- Use case name: ReportEmergency
- Participating actors:
  - Field Officer (Bob and Alice in the Scenario)
  - Dispatcher (John in the Scenario)
- Exceptions:
  - The FieldOfficer is notified immediately if the connection between her terminal and the control room is lost
  - The Dispatcher is notified immediately if the connection between any logged in FieldOfficer and the control room is lost
- Flow of Events: on next slide…
- Special Requirements:
  - The FieldOfficer's report is acknowledged within 30 seconds; the selected response arrives no later than 30 seconds after it is sent by the Dispatcher

Use case example: flow of events

- **Initiation:** The **FieldOfficer** activates the “Report Emergency” function of her terminal. **FRIEND** (the system to be developed) responds by presenting a form to the officer.
- The FieldOfficer fills the form, by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form, at which point, the **Dispatcher** is notified.
- The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the emergency report.
- **Termination condition:** The FieldOfficer receives the acknowledgment and the selected response.

Example of a poor use case

- Use case name: Accident
- Participating Actors: (not an action, i.e., a verb)
  - Field Officer (Dispatcher actor is missing here but mentioned in the next section)
- Flow of Events:
  - 1. The field officer reports the accident
  - 2. An ambulance is dispatched (by whom?)
  - 3. The Dispatcher is notified when the ambulance arrives on site (what does the field officer do after dispatching?)

Tips to define proper use cases

- Use cases named with verbs that indicate what the user is trying to accomplish
- Actors named with nouns
- The boundary of the system should be clear, the steps accomplished by actors and those accomplished by the system should be clearly distinguished
- Use cases' steps in active voice
- The causal relationship between steps should be clear
- A use case per user transaction
- Separate description of exceptions
- Keep use cases small (no more than two/three pages)

Use case example: allocate a resource

- Actors:
  - ***Resource Allocator:*** The Resource Allocator is responsible for allocating resources in case they are scarce
  - ***Dispatcher:*** A Dispatcher updates and removes Emergency Incidents, Actions, and Requests in the system; the Dispatcher also allocates a resource to an Emergency if the resource is available
  - ***Resources:*** The Resources that are allocated to the Emergency

Allocate a resource (1)

- Use case name: AllocateResources
- Participating Actors:
  - Dispatcher (John in the Scenario)
  - Resource Allocator
  - Resources
- Entry Condition
  - An Emergency Incident has been opened
- Flow of Events
  - The Dispatcher selects the types and number of Resources that are needed for the incident
  - FRIEND replies with a list of Resources that fulfill the Dispatcher's request
  - The Dispatcher selects the Resources from the list and allocates them for the incident
  - FRIEND automatically notifies the Resources
  - The Resources send a confirmation

Allocate a resource (2)

- Exit Condition
  - The use case terminates when the resource is committed.
  - The selected Resource is now unavailable to any other Emergency Incidents or Resource Requests
- Exceptions
  - If the list of Resources provided by FRIEND is insufficient to fulfill the needs of the emergency, the Dispatcher informs the Resource Allocator
  - The Resource Allocator analyzes the situation and selects new Resources by decommitting them from their previous work
  - FRIEND automatically notifies the Resources
  - The Resources send a confirmation

Steps when formulating use cases

- First step: name the use case
  - Use case name: ReportEmergency
- Second step: Find the actors
  - Generalize the concrete names (“Bob”) to participating actors (“Field officer”)
  - Participating Actors:
    - Field Officer (Bob and Alice in the Scenario)
    - Dispatcher (John in the Scenario)
- Third step: Then concentrate on the flow of events
  - Use informal natural language

How to specify a use case (summary)

- Name of Use Case
- Actors
  - Description of Actors involved in use case
- Entry condition
  - “This use case starts when…”
- Flow of Events
  - Free form, informal natural language
- Exit condition
  - “This use case terminates when…”
- Exceptions
  - Describe what happens if things go wrong
- Special Requirements
  - Nonfunctional Requirements, Constraints

Use cases to functional requirements

- Each use case may lead to a requirement
- Examples
  - FRIEND shall support Field Officers in reporting an emergence
  - FRIEND shall support Dispatchers in allocating the resources to the incident
- The corresponding use cases will then describe in detail how the requirements are fulfilled

Politecnico
di Milano

Modeling requirements

Modeling requirements

- What is a model
- What to model in RE
- Tools for modeling

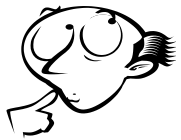
The Central Role of Requirements Models

stakeholders

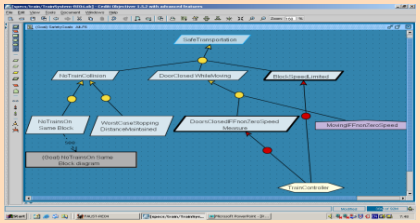
existing systems

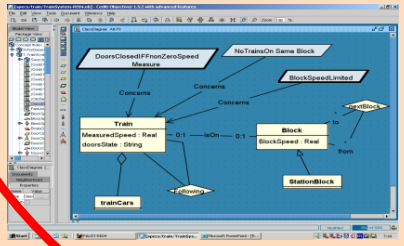
documents

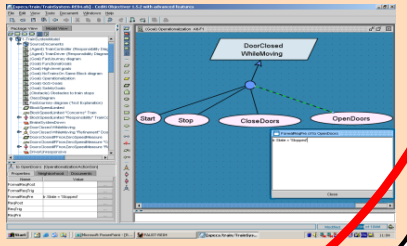
elicitation
& **modelling**



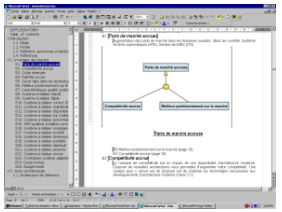
Requirements Models







generation of
RE deliverables



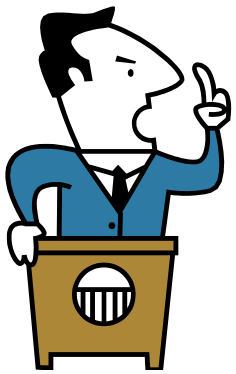
requirements document

analysis
& validation

Model: A definition

"A model is a representation in a certain medium of something in the same or another medium.
The model captures the important aspects of the thing being modeled and simplifies or omits the rest"

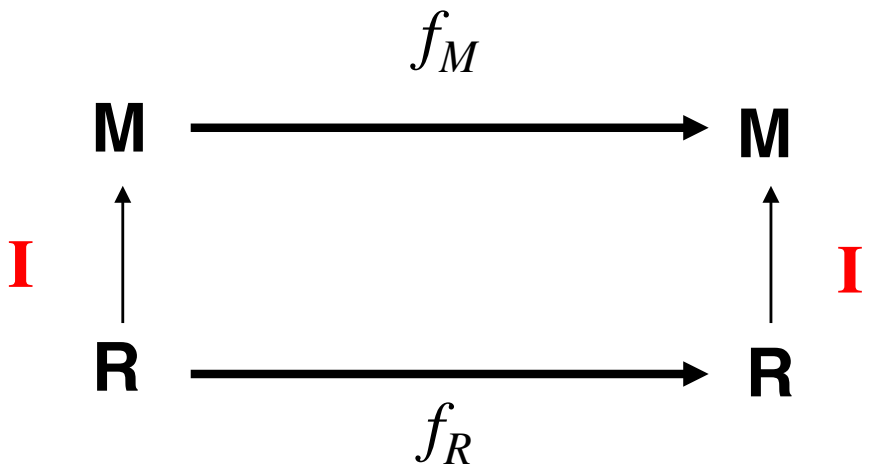
Grady Booch



Reality and Model

- Reality R: Real Things, People, Processes, Relationship
- Model M: Abstractions from (really existing or only thought of) things, people, processes and relationships between these abstractions

What is a "good" model?



- I is the mapping of real things in reality R to abstractions in the model M
  - also called **Interpretation**
- Relationships, which are valid in reality R, are also valid in model M
  - f_R: relationship between real things in R
  - f_M: relationship between abstractions in M

Why models?

- We use models
  - To abstract away from details in the reality, so we can draw complicated conclusions in the reality with simple steps in the model
  - To get insights into the past or present
  - To make predictions about the future

Models of software systems

- A model of a software system is a representation of the system from a specific point of view
  - expressed in a modelling language
  - has syntax and semantics
  - easier to use for a specific purpose than the final system
- Modeling language
  - artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules.

What are software models for

- Capture and precisely state requirements and domain knowledge
- Think about the design of a software system
- Generate usable work products
- Give a simplified view of complex systems
- Evaluate and simulate a complex system
- Generate potential configurations of systems
  - all consistent configurations should be possible
  - not always possible to represent all constraints in the model (model is an abstraction !)

Modelling issues

- Coherence
  - different views of the system must be coherent
- Variations in interpretation and ambiguity
  - define where different interpretations of the model are acceptable

What should we model in RE?

- The objects and people that are of interest for the given problem
  - E.g., the aircraft and the sensors and actuators relevant to the braking system
- The relevant phenomena
  - Weels_turning, Reverse_enabled, …
- The goals, requirements, and domain assumptions

Which tools can we use for modeling?

- Any language (e.g., Italian, English, …)
  - Pros: simplicity of use
  - Cons:
    - high level of ambiguity,
    - it is easy to forget to include relevant information
- A formal language (e.g., first order logic, Alloy, Z, …)
  - Pros:
    - possibility to use some tool to support analysis and validation
    - the approach forces the user in specifying all relevant details
  - Cons: you need to be expert in the use of the language

Which tools can we use for modeling?

- A semi-formal language like UML
  - Pros:
    - simpler than a formal language
    - imposes some kind of structure in the models
  - Cons:
    - not amenable for automated analysis
    - some level of ambiguity
- A mixed approach
  - Use a semi-formal language for the basics
  - Comment and complement the semi-formal models with explanatory informal text
  - Use a formal language for the most critical parts

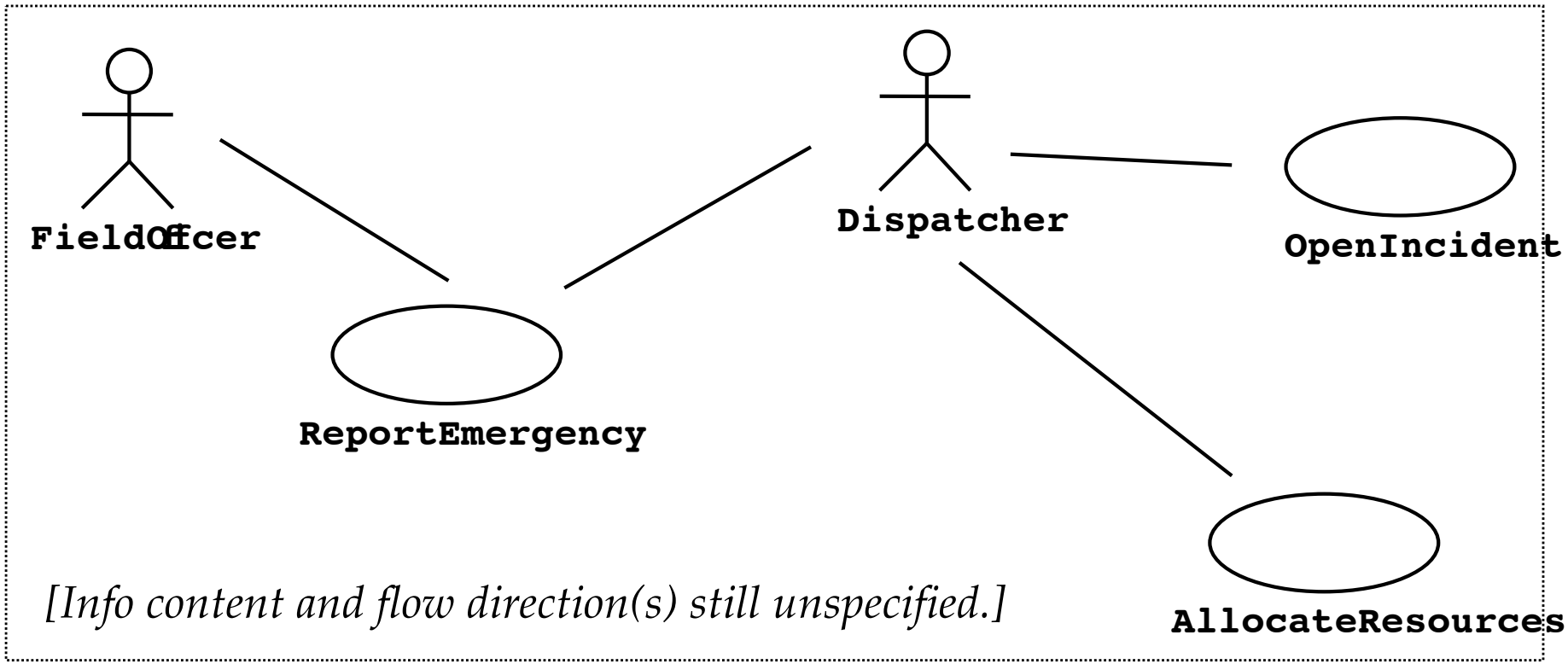
UML and requirements engineering

Use Cases (reprise)

- A use case is a flow of events in the system, including interaction with actors
- It is initiated by an actor
- Each use case has a name
- Each use case has a termination condition

Use Case Model: The set of all use cases specifying the complete functionality of the system

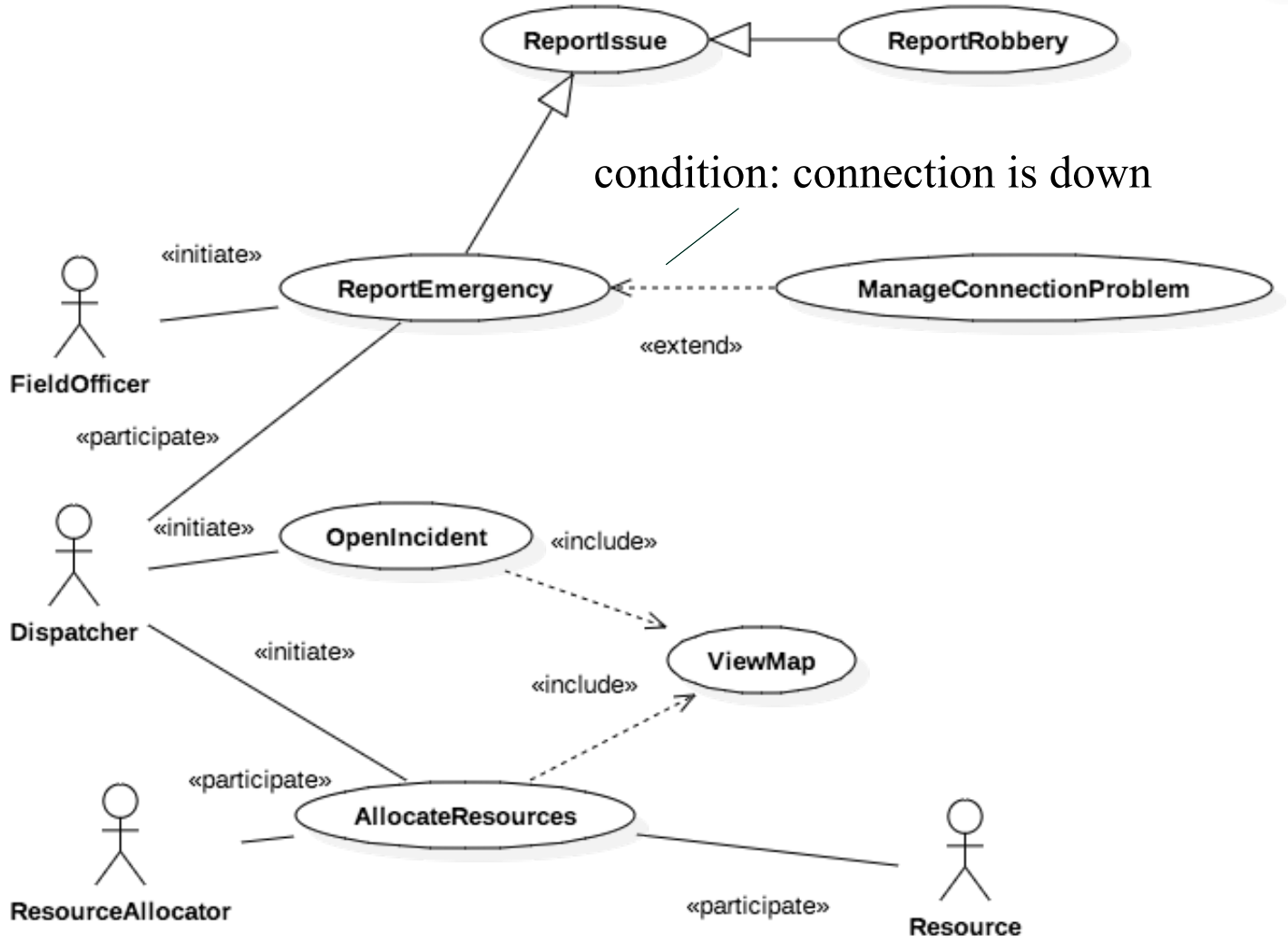
Example: Use Case Model for Incident Management



Use Case Associations

- A use case association is a relationship between use cases
- Important types of use case associations
  - Include
    - A use case uses another use case (“functional decomposition”)
  - Extends
    - A use case extends another use case
  - Generalization
    - An abstract use case has several different specializations

Examples


ReportIssue
ReportRobbery
condition: connection is down
«initiate»
ReportEmergency
ManageConnectionProblem
«extend»
FieldOfficer
«participate»
«initiate»
OpenIncident
«include»
Dispatcher
«initiate»
ViewMap
«include»
«participate»
AllocateResources
ResourceAllocator
«participate»
Resource

Requirements-level class diagrams

- They are conceptual models for the application domain
  - different from OO software design models
- They may model objects that will not be represented in the software-to-be
  - because this is not known at the start of the requirements modelling effort
- Usually, they do not attach operations (methods) to objects
  - it's best to postpone this kind of decisions until software design

How do you find objects and classes?

- Analyze any description of the problem and application domain you may have
- Analyze your scenarios and user cases descriptions
- Finding objects is the central piece in object modeling
- A possible tool to use in the analysis
  - Abbott Textual Analysis, 1983, also called noun-verb analysis
    - Nouns are good candidates for classes
    - Verbs are good candidates for associations and operations

Object modeling: an Example

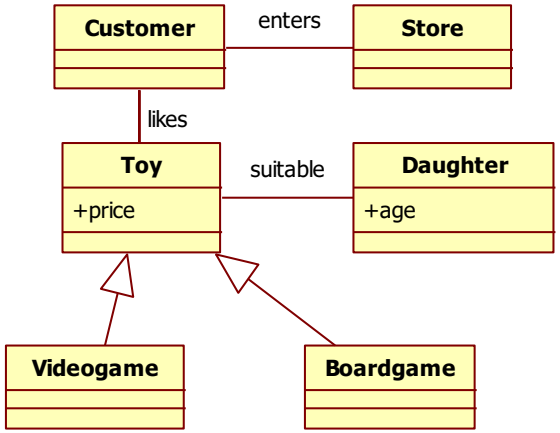
- Consider this text
  - *The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.*
  - *An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely Monopoly (a boardgame). The customer buys the game and leaves the store*

Textual Analysis using Abbot's technique of Natural Language Analysis [OOSE 5.4.1]

Example	*Grammatical construct*	*UML Component*
"Monopoly"	Concrete Person, Thing	Object
"toy"	noun	Class
"boardgame"	noun	Class
"3 years old"	Adjective	Attribute
"enters"	verb	Operation/Association
"depends on …"	Intransitive verb	Association
"is a", "either … or", "kind of …"	Classifying verb	Inheritance
"Has a", "consists of"	Possessive Verb	Aggregation
"must be", "less than …"	modal Verb	Constraint

Generation of the class diagram for the example

- The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.
- An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely Monopoly (a boardgame). The customer buys the game and leaves the store



Finding Participating Objects in Use Cases

- Pick a use case and look at its flow of events
  - Look for recurring nouns (e.g., Incident),
  - Identify real-world entities that the system needs to keep track of (e.g., FieldOfficer, Dispatcher, Resource),
  - Identify real-world procedures that the system needs to keep track of (e.g., EmergencyOperationsPlan),
  - Identify data sources or sinks (e.g., Printer)
  - Identify interface artifacts (e.g., PoliceStation) [e.g., telecomm link?]
- Be prepared that some objects are still missing and need to be found
- Always use the user's terms

Object modeling and the Report Emergency use case

- Nouns are in red
- Verbs are in green

- The FieldOfficer activates the “Report Emergency” function of her terminal. FRIEND responds by presenting a form to the officer.
- The FieldOfficer fills the form, by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form, at which point, the Dispatcher is notified.
- The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the emergency report.
- The FieldOfficer receives the acknowledgment and the selected response.

Textual analysis

Term	Grammatical construct	UML Component
FieldOfficer	Noun	Class
FRIEND	Noun	This is the S2B object of our analysis. It does not have a corresponding Class.
Emergency	Noun	Class
Level, type, location	Noun	Attributes
Brief description	Noun + Adjective	Attribute
Response	Noun	Class
Dispatcher	Noun	Class
Incident	Noun synonym of emergency	No need for a new class
Emergency report	Noun	Class containing the attributes listed above

Textual analysis

Term	Grammatical construct	UML Component
Activates	Verb	Association/Operation
Responds	Verb	
Fills	Verb	Association/Operation
Describes	Verb	Association/Operation
Submits	Verb	Association/Operation
Is notified	Passive verb	
Creates	Verb	Association/Operation
Selects	Verb	Association/Operation
Acknowledges	Verb	Association/Operation
Receives	Verb	Event

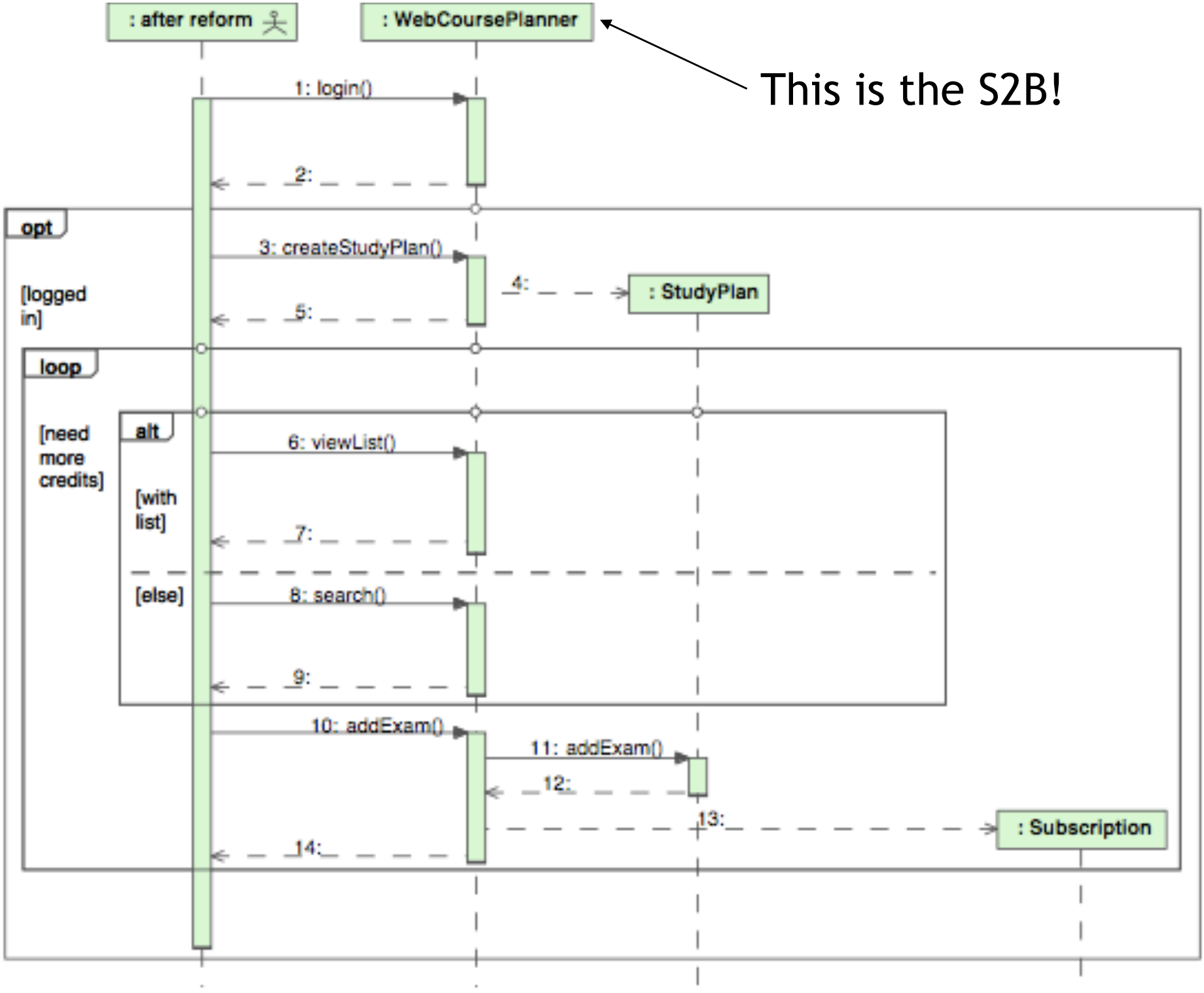
Now you can try to define the class diagram yourself

Dynamic modeling

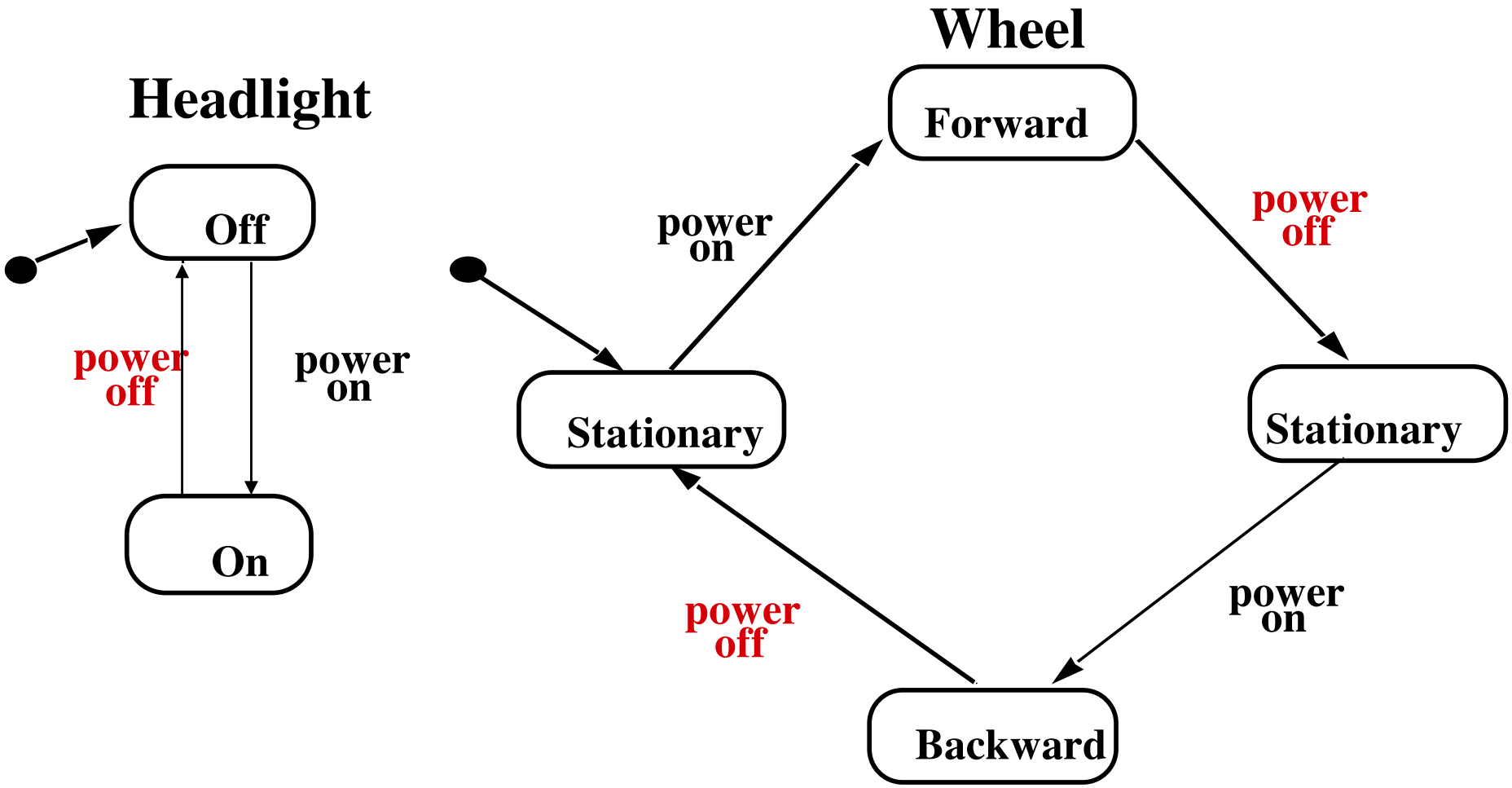
- Purpose:
  - Supply methods to model interactions, behaviors of participants and workflow
- How do we do this?
  - Start with use case or scenario
    - Model interaction between objects => sequence diagram
  - Model dynamic behavior of a single object => statechart diagram
  - Model workflow => activity diagram

Sequence diagram

- From the flow of events in the use case or scenario, proceed to the sequence diagram
- A sequence diagram is a graphical description of objects participating in a use case or scenario using a DAG (directed acyclic graph) notation
- Relation to object identification:
  - Objects/classes have already been identified during object modeling
  - Other objects are identified as a result of dynamic modeling
- Heuristic:
  - An event always has a sender and a receiver
  - The representation of the event is sometimes called a message
  - Find sender and receiver for each event => These are the objects participating in the use case


: after reform
: WebCoursePlanner
This is the S2B!
1: login()
2:
opt
3: createStudyPlan()
4:
: StudyPlan
[logged in]
5:
loop
[need more credits]
alt
6: viewList()
[with list]
7:
[else]
8: search()
9:
10: addExam()
11: addExam()
12:
13:
: Subscription
14:

An example of statechart: Toy car

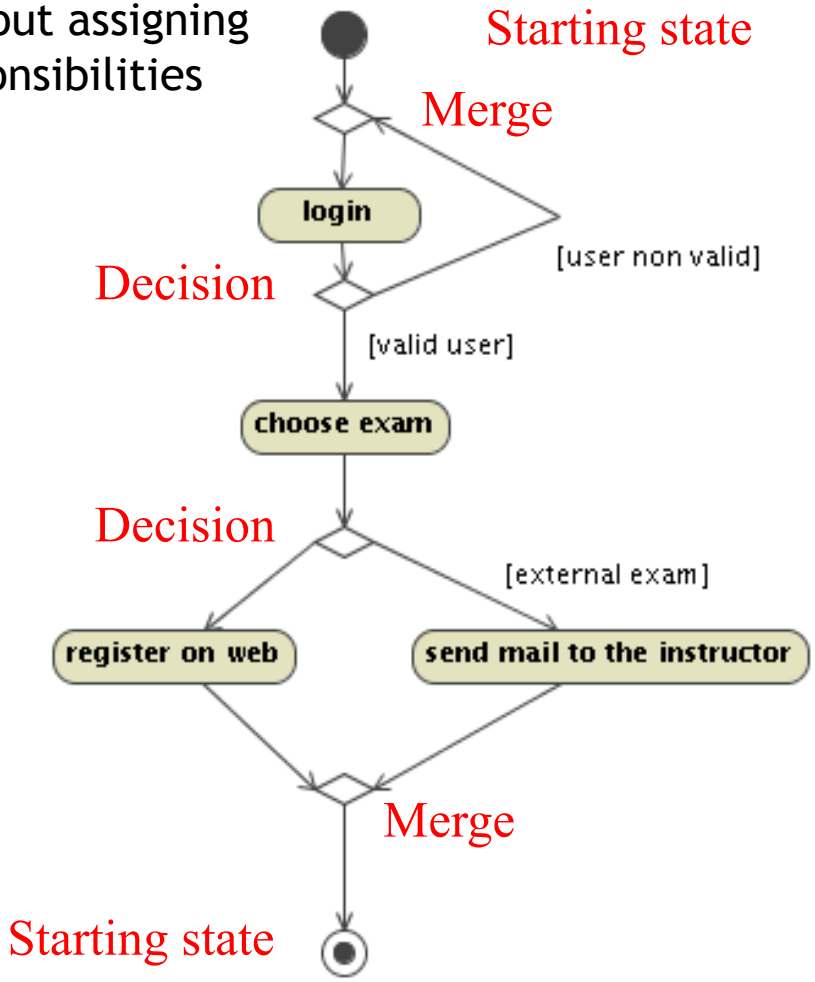


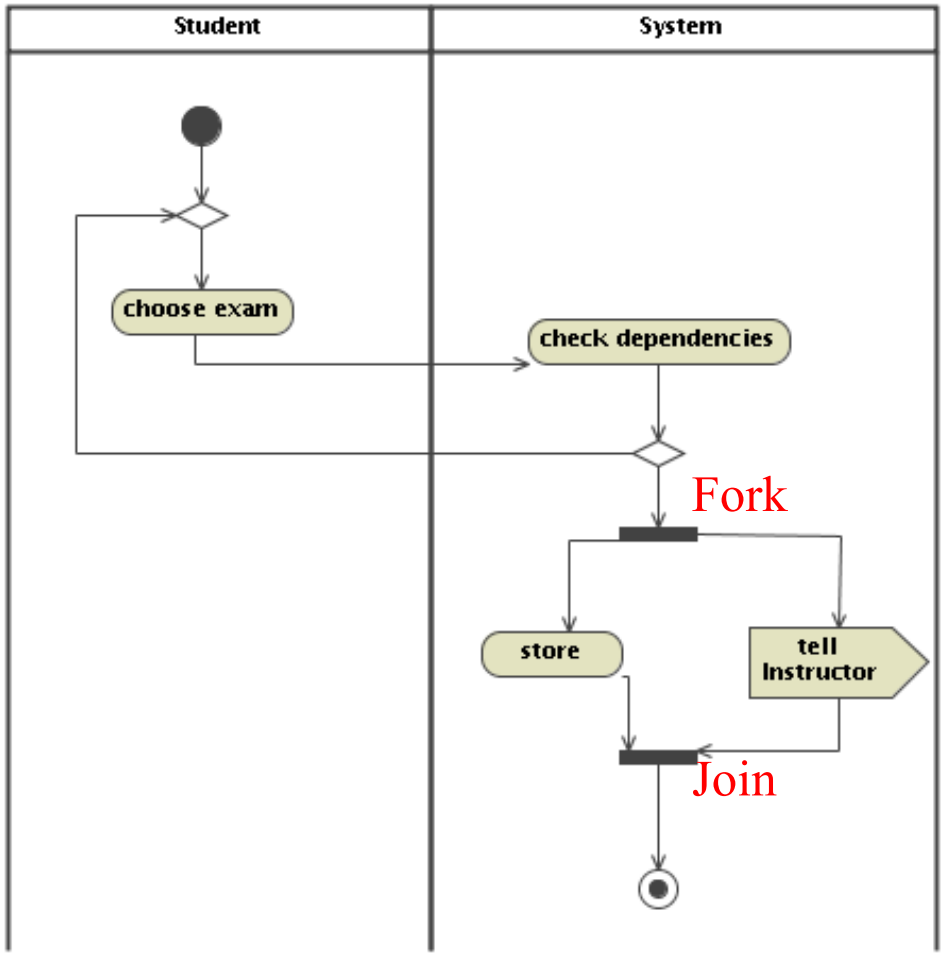
Statechart vs sequence diagram

Statechart	Sequence diagram
Focus on changes in an individual object over time	Focus on the interaction between objects over time
Class-level documentation	Instance-level documentation
We can infer many possible event and order-dependent behaviors	Shows one specific case or a subset of cases

Activity diagrams

Describes an activity without assigning responsibilities





Describes “register on the web” by highlighting the responsibilities of the student and of the system

Sequence diagram vs activity diagram

Sequence diagram	Activity diagram
Focus on object interaction	Focus on activities and flow of activities
Suited for describing an interaction protocol	Suited for describing a process

Summary: Requirements analysis

- What are the transformations?
  - Create scenarios and use case diagrams
    - Talk to client, observe, get historical records, do thought experiments

Functional Modeling

- What is the structure of the world?
  - Create class diagrams [static information models]
    - Identify objects
    - What are the associations between them?
    - What is their multiplicity?
    - What are the attributes of the objects?
    - What operations are defined on the objects?
  - Is there any state change in an object that is to be defined explicitly? If yes, create statecharts [dynamic class behavior models]

Object Modeling

Summary: Requirements analysis (cont)

- How is the expected interaction between the S2B and the environment?

Dynamic Modeling

  - Create sequence diagrams from use cases [dynamic object behavior instance examples]
    - Identify event senders and receivers
    - Show sequence of events exchanged between objects Identify event dependencies and event concurrency
  - Create activity diagrams when you want to highlight important processes [workflow]