

# Domain Analysis

Gerson Sunyé  
University of Nantes  
[gerson.sunye@univ-nantes.fr](mailto:gerson.sunye@univ-nantes.fr)  
<http://sunye.free.fr>

# Agenda

- Domain analysis definition
- Basic concepts
- Analysis process
- Analysis Patterns and Best practices
- Effective Use Cases

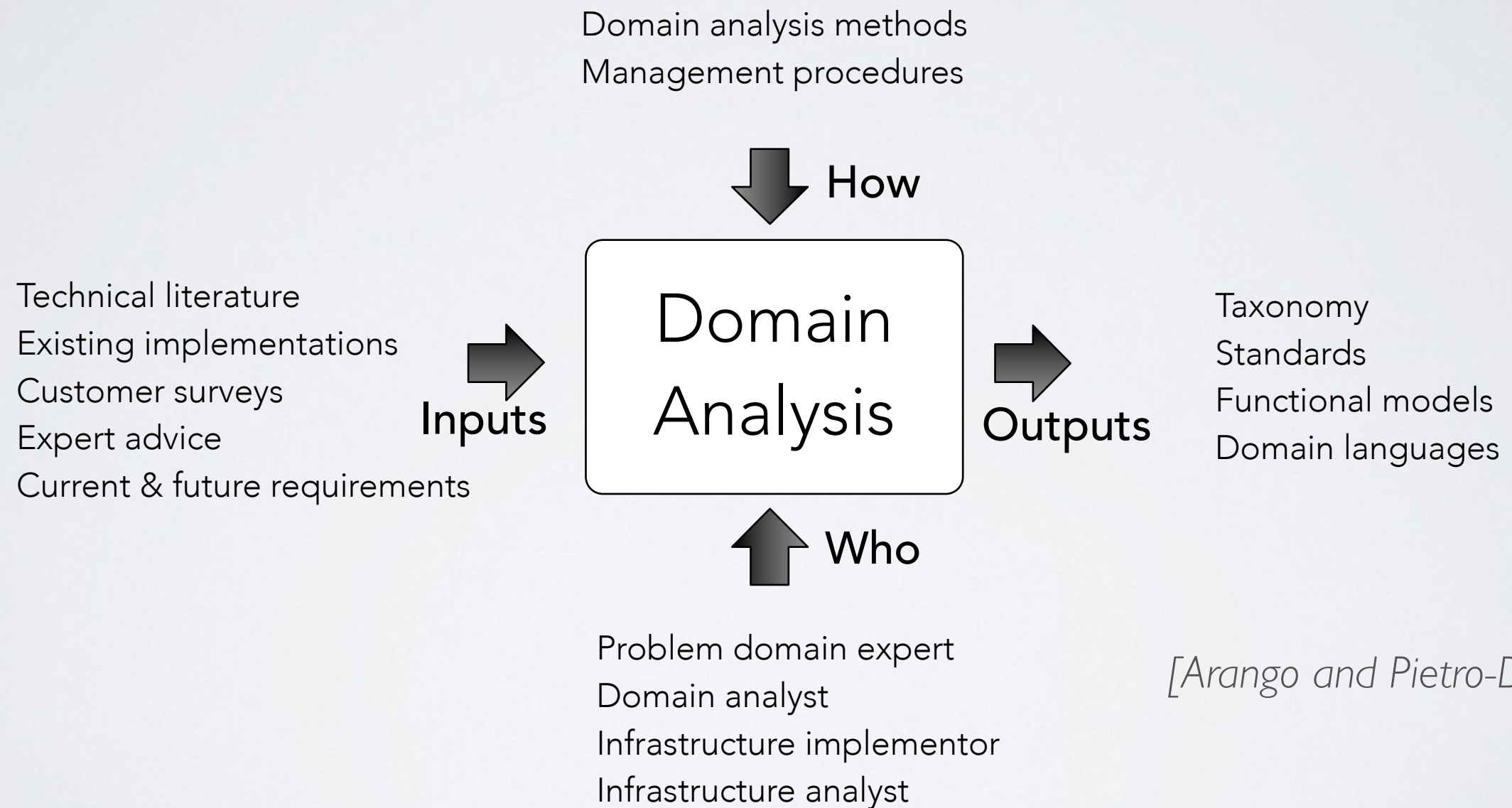
# Domain Analysis

# Definition

- Domain analysis is the process of identifying, collecting, organizing, analyzing and representing a domain model from the study of existing systems, underlying theory, emerging technology and development histories within the domain of interest.

*The Free On-line Dictionary of Computing, © Denis Howe*

# The Big Picture



*[Arango and Pietro-Diaz 1989]*



# Goals

- Understand the problem domain.
- Identify actors, activities, rules, and the domain vocabulary.
- Provide a base for different designs.
- Create reusable resources, across different projects.
- Support the creation of domain-specific languages.

# Approaches

- Model the current system “as is”.
- Improve the current system and model the system “to be”.

# Domain Model

- Represents meaningful conceptual classes in a problem domain.
- Represents real-world concepts (the problem), not software components (the solution).
- May be shared by different projects.

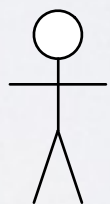


# Basic Concepts

# Domain Objects

- An object is composed of information and behavior.
- Similar objects are typed by a Conceptual Class.

113 : Classroom



John : Teacher

# Actions

- Actions are something that happen in the system: events, tasks, jobs, changes of state, activities, etc.
- They are or may be independent from objects, and associated to objects later.
- Actions modify objects.
- Actions are occurrences of Use Cases.

# Objects and Actions

- Objects and Actions are at the same level.
- Creating a sound domain model requires careful thought on actions and what they achieve.

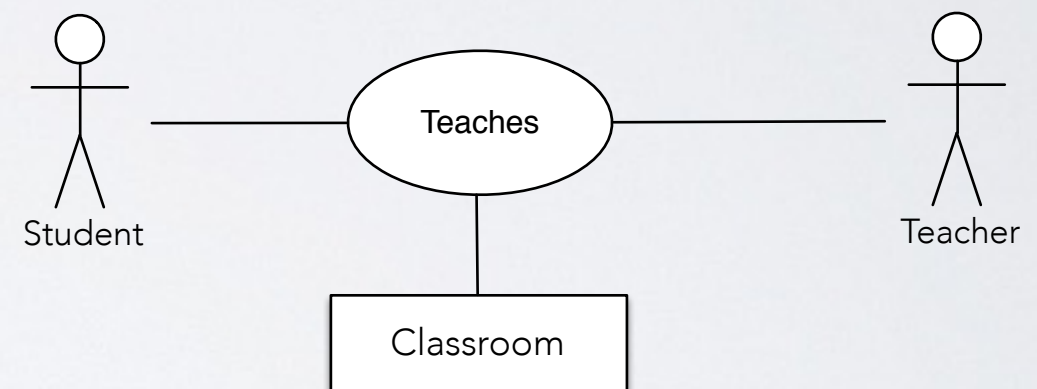
# Fractals

- A fractal picture has the same appearance at all scales.
- Objects: business departments, machines, running software components, programming language concepts.
- Actions: interactions among objects: big business deals, phone calls, bike rides, etc.



# Actions represent Interactions

- Actions represent interactions between objects.
- An interaction (Sequence, Communication Diagrams) describes how real-world objects interact.



# Objects participate in Actions

- Actions have participants, objects.
- How many objects are needed?  
Enough to describe the actions.
- Associations provide a vocabulary in which it is possible to describe effects of actions.

# Actions affect Objects

- Objects are used to describe action's effects on participants
- Actions characterized by what they achieve (its effects) and specified by a post-condition.
- Post-conditions may introduce new terms to describe the effect.

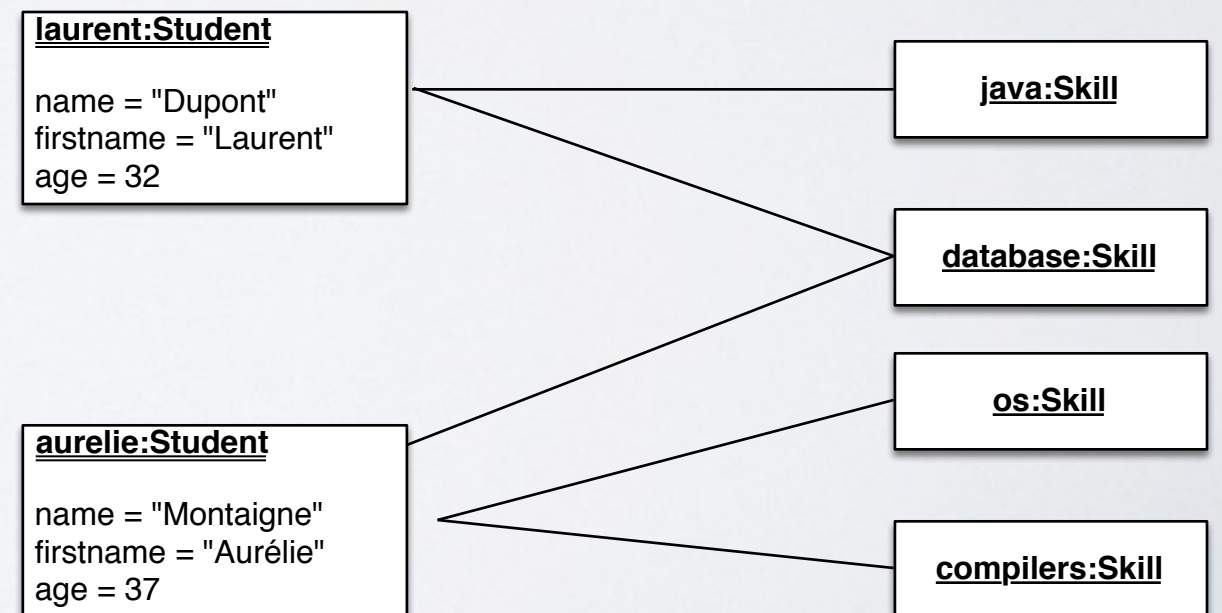
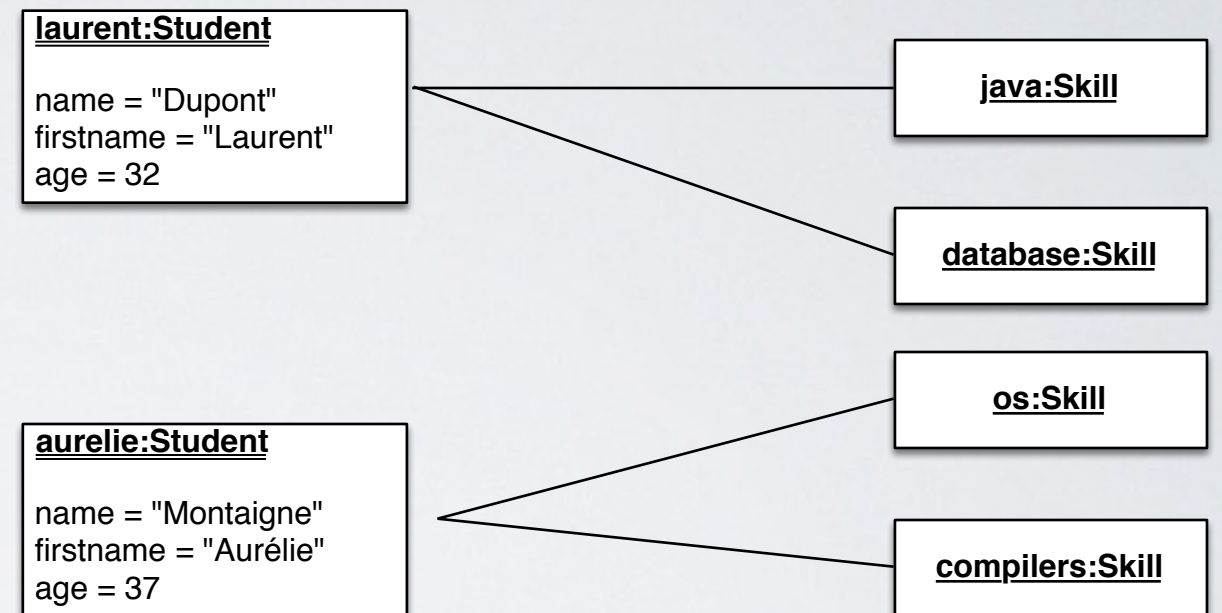
```
UC::teach(student, teacher, skill)
```

```
post:
```

```
– this skill has been added to the student's accomplishments
```

# Snapshots describe Actions

- Object diagrams may represent the effects of an action.
- Here, the effects of:  
`teach(aurelie, paul, database)`





# Post-conditions

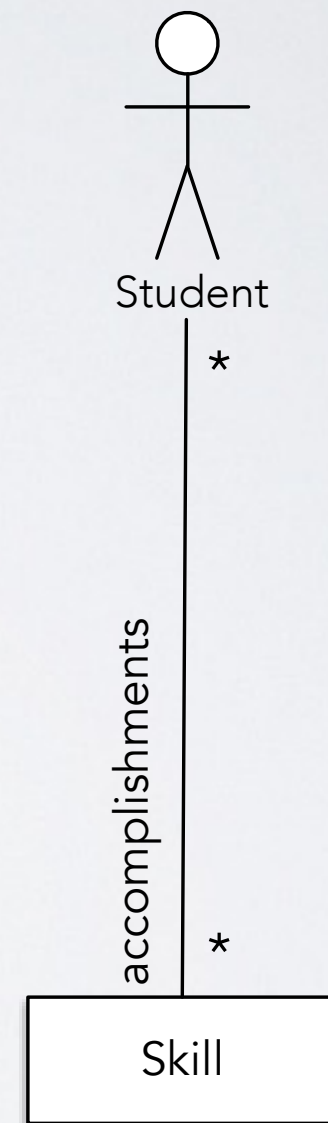
- A post condition is a relationship between the before and after states of an object or a set of objects
- Post conditions make precise statements about what happens in an action, even at an abstract level.

```
UC::teach(student, teacher, skill)
post:
    --this skill has been added to the student's accomplishments
student.accomplishments =
    student.accomplishments@pre->including(skill).
```



# Post-conditions introduce new terms

- New terms: associations, attributes, or objects.



```
UC::teach(student, teacher, skill)
```

```
post:
```

```
--this skill has been added to the student's accomplishments
student.accomplishments =
    student.accomplishments@pre->including(skill).
```

# Types of Actions

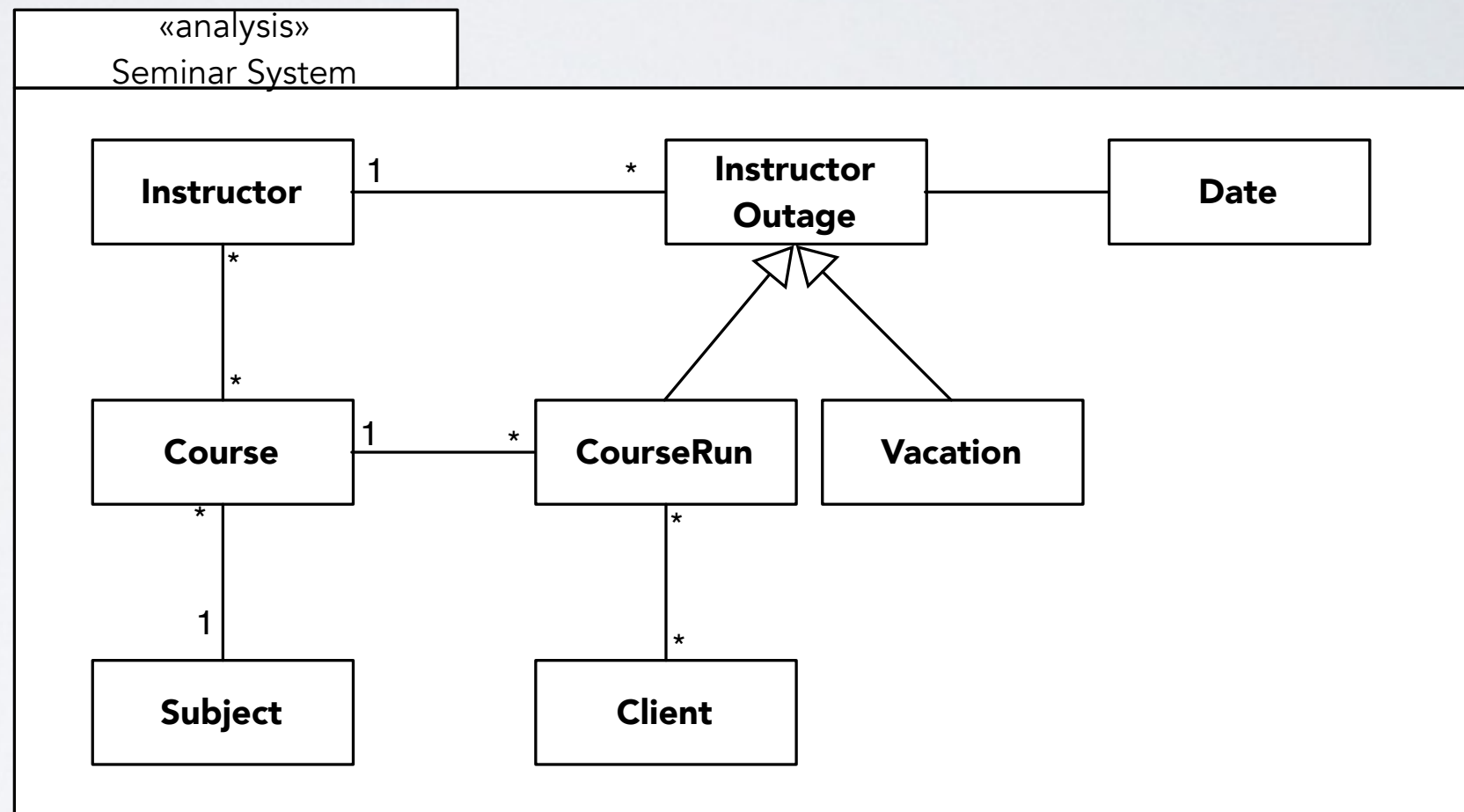
- Use Case
  - Not-oriented, symmetric, with N participants.
  - Also called “Joint Action”.
- Operations
  - Oriented: one sender, one receiver.

# Associations

- Associations represent real world relationships, not necessarily logical ones.
- Associations just exist.
- The use of associations provides:
  - A vocabulary for the actions (the static model).
  - The need objects for a given action (the dynamic model).

# Domain Model

- Represents conceptual classes, associations and attributes.
- Uses a simplified UML Class Diagram Notation.





# Static Invariants

- A static invariant is a condition that should always be true, at least between the execution of any action.
- They are written by following the links from a given starting point.
- They are described informally and written in a simple language of boolean expressions (OCL).



# Static Invariant Example

The analyst has decided that vacation and course run are both kinds of instructor outage (instructor not available)

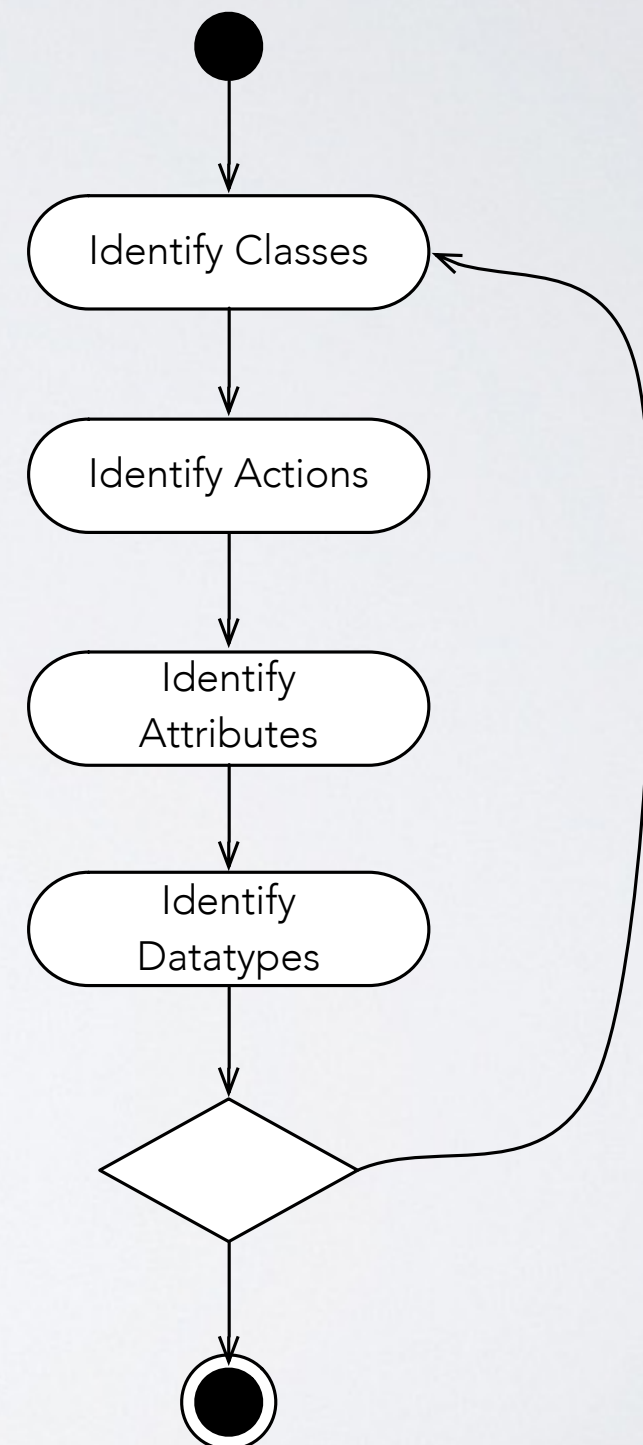
```
inv -- for every CourseRun, its instructor's  
qualifications must include the course  
context CourseRun
```

```
inv: instructor.qualifications->includes(course)
```

# Analysis Process

# Steps

1. Identify Conceptual Classes.
2. Identify Actions.
3. Identify Attributes.
4. Identify Datatypes.



# Identifying Conceptual Classes

- Identify Nouns and Noun Phrases in descriptions:
  - Existing procedures, standards documents, software, and user-manuals.
  - Observation and interviews.
- Existing models: relational, ER, etc.
- Existing batch-mode systems.
- Feedback from prototypes, scenarios, models, storyboards, etc.



# Common Candidates

- Tangible objects, Descriptions, Roles, Places, Transactions, Containers, Systems, Abstract nouns, Rules, Organizations, Events, Processes, Written Materials, Catalogs, Records, Financial Instruments, and Services



# Filtering

- Different phrases may represent the same concept: remove redundant concepts (synonyms).
- Remove concepts that are ambiguous, vague, irrelevant.

# Identifying Actions

- For each conceptual class identified previously, ask the following questions:
  - What do you do?
  - Whom do you deal with?
- Every time a new verb appears, a new action is identified.

# Relating Actions and Classes

- Actions are not assigned to objects from the beginning.
- Steps:
  1. What happens.
  2. Which object is responsible for initiating what happens.
  3. How it is done.

# Finding new Classes

- For each action identified, ask:
  - Who participates in this action?
  - What is the impact on their state?
- Answers to these questions lead to new conceptual classes and to new post conditions.



# Identifying Associations

- New conceptual classes and post conditions lead to associations and attributes, which help to illustrate the action effects.
- For each new conceptual class identified, ask:
  - What action affect it?
  - What actions does it affect?
- Traverse up and down the abstraction tree.

# Identifying Attributes

- New conceptual classes and postconditions also lead to new attributes.
- For each new conceptual class identified, ask:
  - How can it be identified?
  - What properties are related to it?
- Attributes are logical properties of objects.
- Attributes should be simple.

# Identify Datatypes

- Use Datatypes instead of Primitive types, such as String or Integer, if:
  - It has several parts/sections (phone number, name, etc.).
  - It has simple operations, such as parsing and validation (credit card number, social security number, etc.).
  - It has other attributes (a promotional price with start and end dates).

# Identify Datatypes (2)

- Use Datatypes instead of Primitive types, such as String or Integer, if:
  - It has a quantity and a unit (a duration has a unit of time).
  - It is widely used as a domain identifier:
    - International Standard Book Number (ISBN).
    - European Article Number (EAN).



# Analysis Best Practices and Patterns

# Use a Simplified UML Syntax

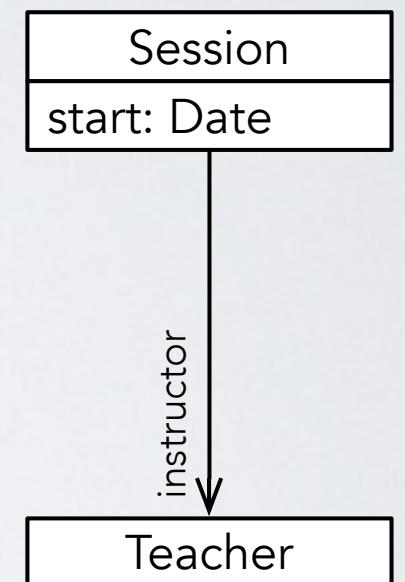
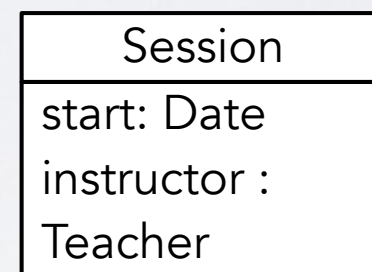
- Use an adapted subset of the UML syntax, some concepts are not meant to be used in domain analysis: visibility, active classes, etc.
- Use specializations to avoid duplicate attributes and associations.
- Name association roles accordingly and use precise cardinalities.
- Avoid the complexity of ternary and quaternary associations.
- Be attentive to class, attribute and role names.

# Be Coherent

- Sequence diagrams (scenarios) are always attached to Use Cases.
- Each Use Case should be described by a main and several alternative or exception scenarios.
- When an object receives a message, its class must defined the operation corresponding to that message.
- State machines are always attached to Classes. Attributes an Operations used by states and transitions must be defined in the corresponding Class.
- Each transition should be described by a scenario.

# Distinguish Attributes and Associations

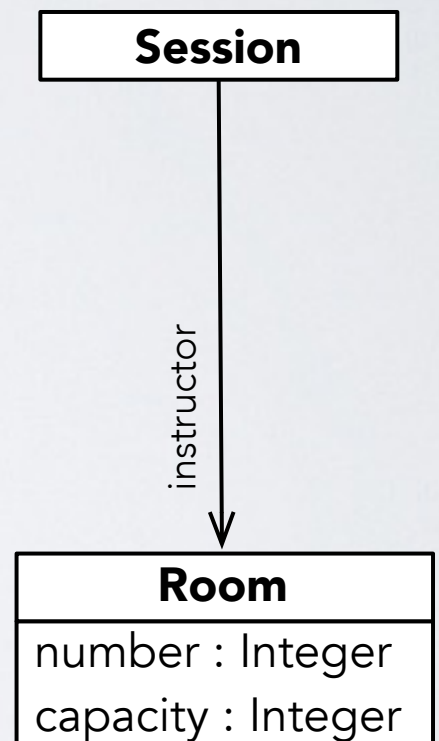
- Use associations to relate two conceptual classes.
- If the attribute type is a data type, it may be shown in the attribute box.





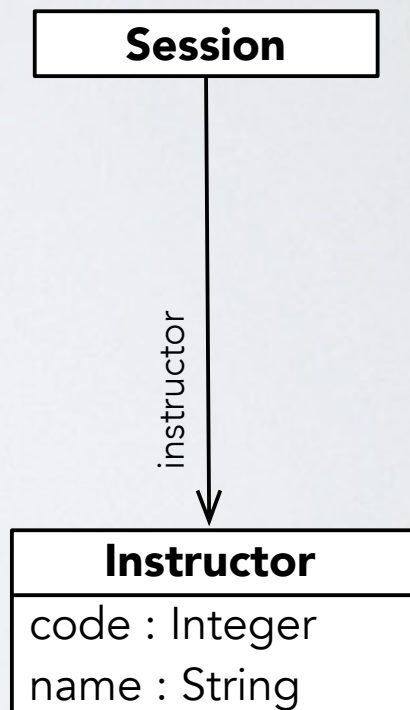
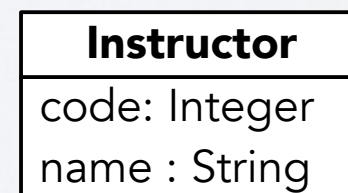
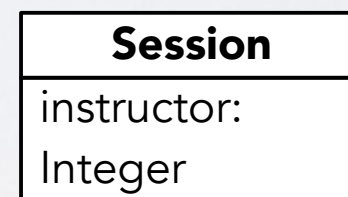
# Avoid representing Complex Concepts as Attributes.

- If the attribute has other attributes, it is a conceptual class.



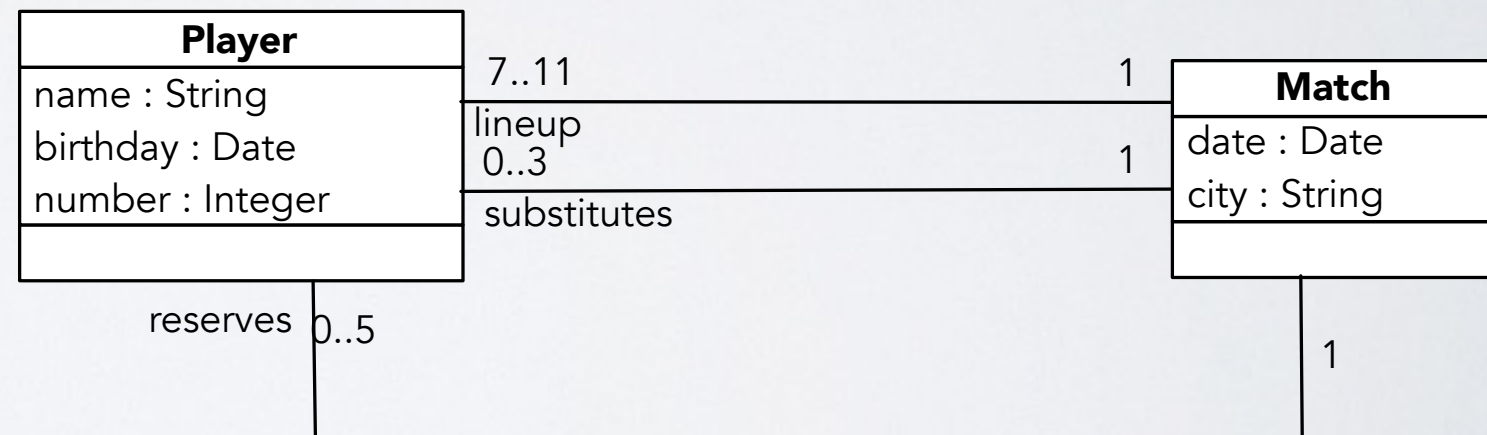
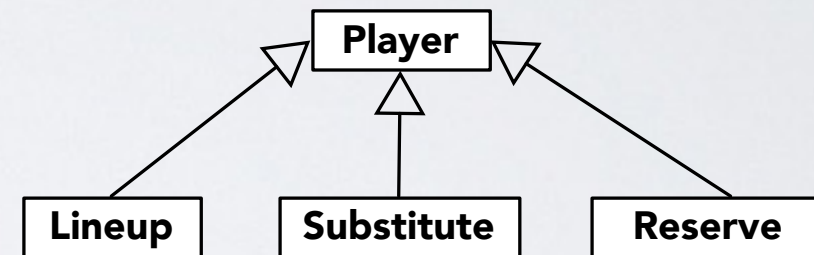
# Avoid Attributes as Foreign Keys.

- Relate concept classes directly.



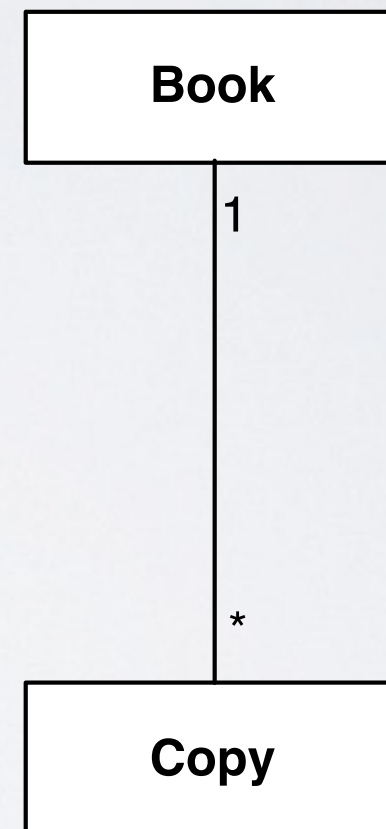
# Distinguish Roles and Classes

- If the concept type depends on other concepts, consider using roles.
- Roles of a Professor on a given Course: instructor, grader, course builder, teaching assistant.



# Items and Descriptors

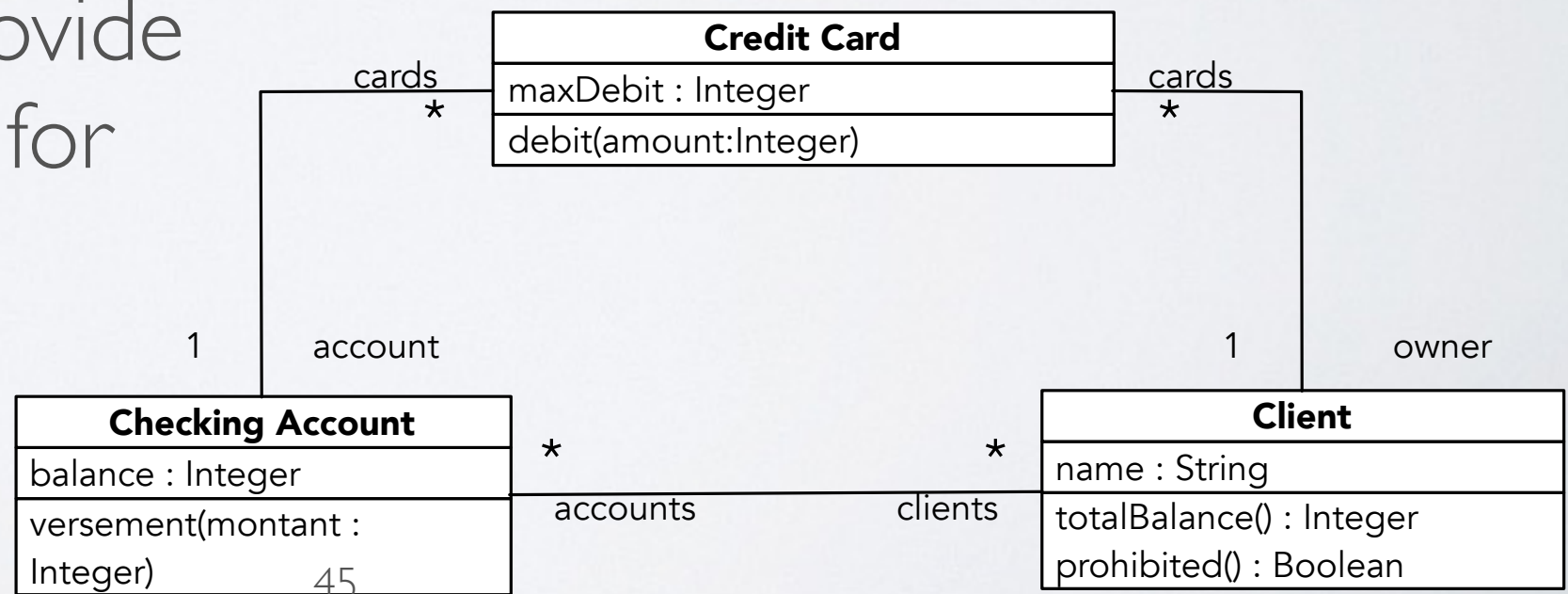
- Distinguish items and their descriptors:
  - Cars
  - Flights
  - Films and DVD.





# Invariants from Association Loops

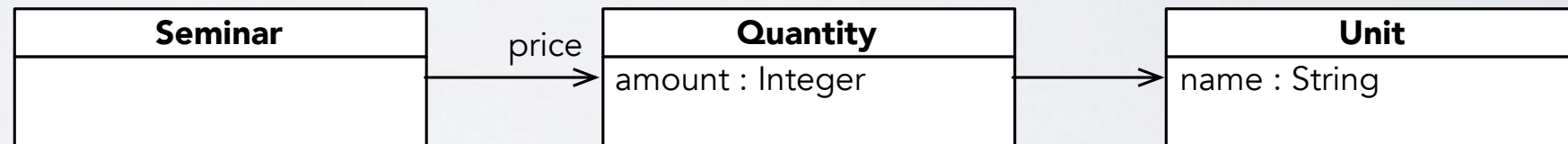
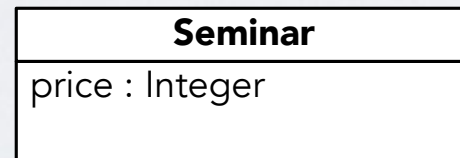
- An invariant is a condition that should always be true.
- Association loops provide an excellent context for invariants.



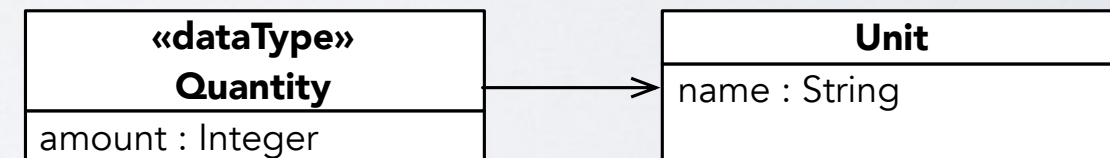
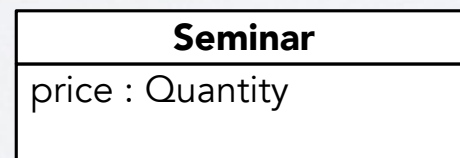
# Quantities and Units

1. Primitive types

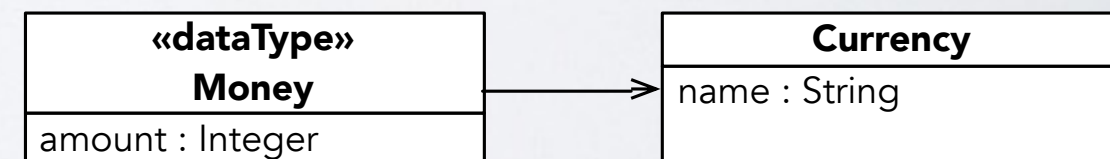
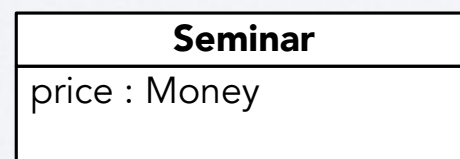
2. Separate quantities and units.



3. Quantity as a data type



4. Money as a specialization of Quantity.



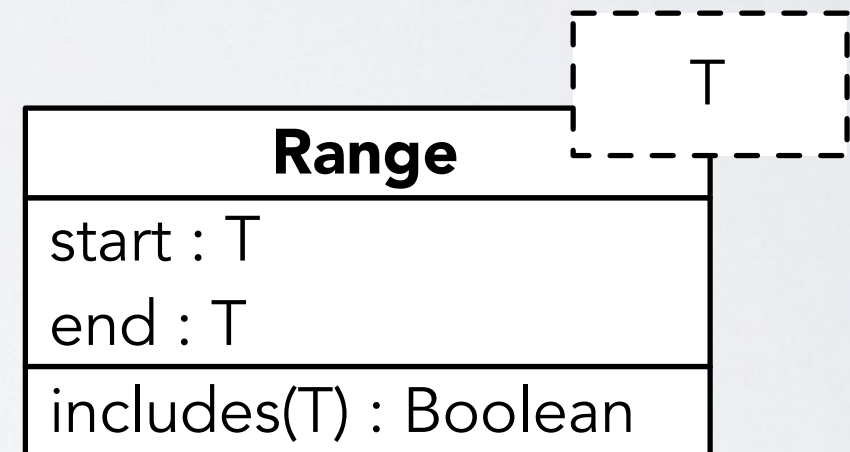
# Quantities and Units (2)

- Enrich Quantity with dedicated operations.

Quantity
amount : Real unit : Unit
+, -, *, / : Quantity <, >, = : Boolean to(Unit) : Quantity toString() : String parse(String) : Quantity

# Range

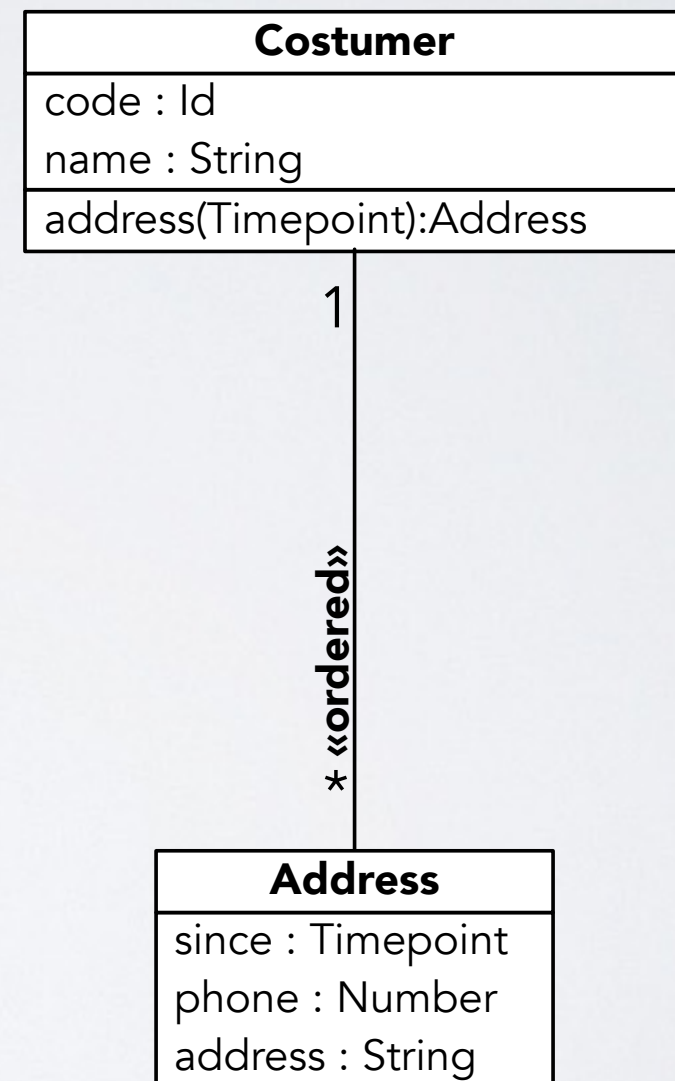
- Range of values as a single object.
- Oct 22-25.



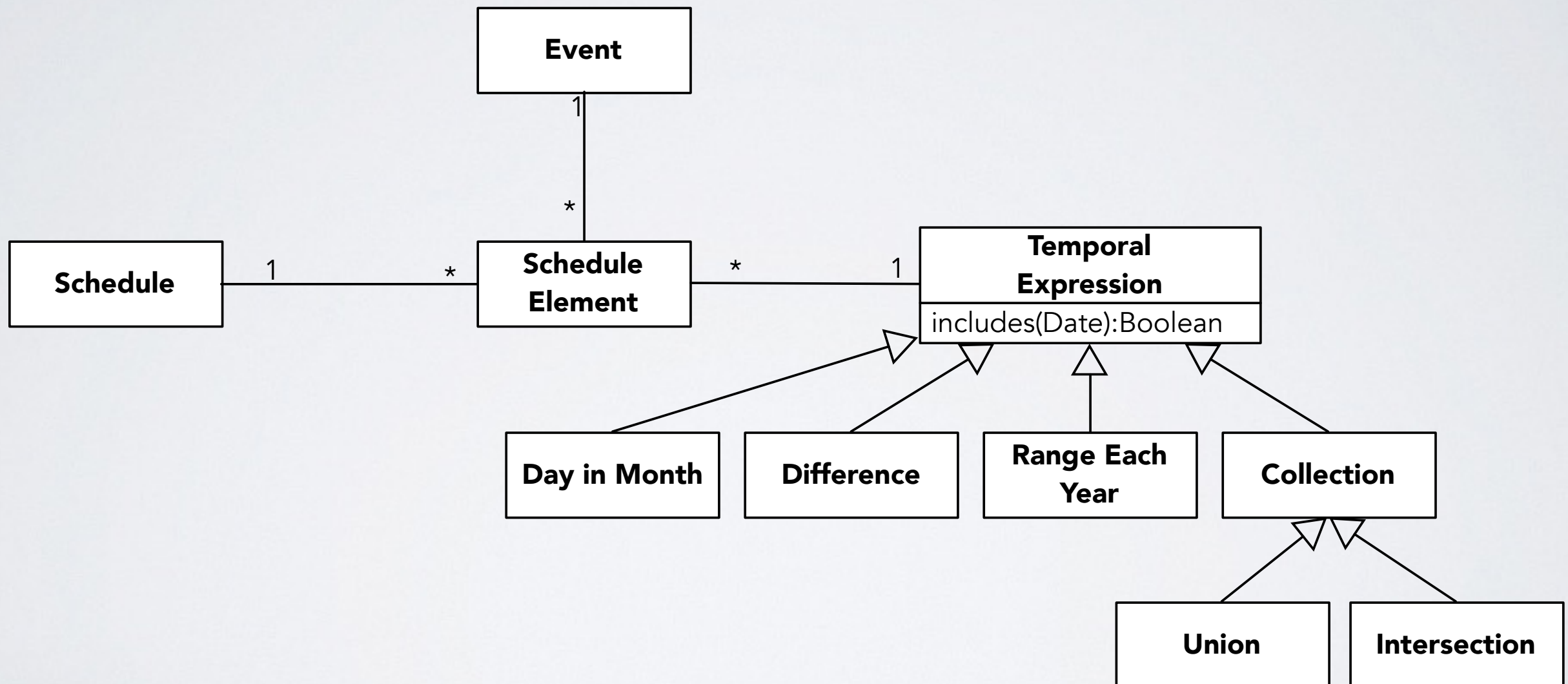


# Temporal Properties

- Modeling concepts that change with time:
- Contract and Contract Version.
- Marriage.



# Recurring Events



# Other Modeling Techniques

# Other Modeling Techniques

- Data Dictionary.
- Concept Maps.
- Domain Specific Languages (DSL).
- UML.



# Data Dictionary

- Centralized repository of information about data.
- Defines the domain terminology.
- Entry point for analysts, designers and developers.
- Informal, scalable and simple.

# Data Dictionary (2)

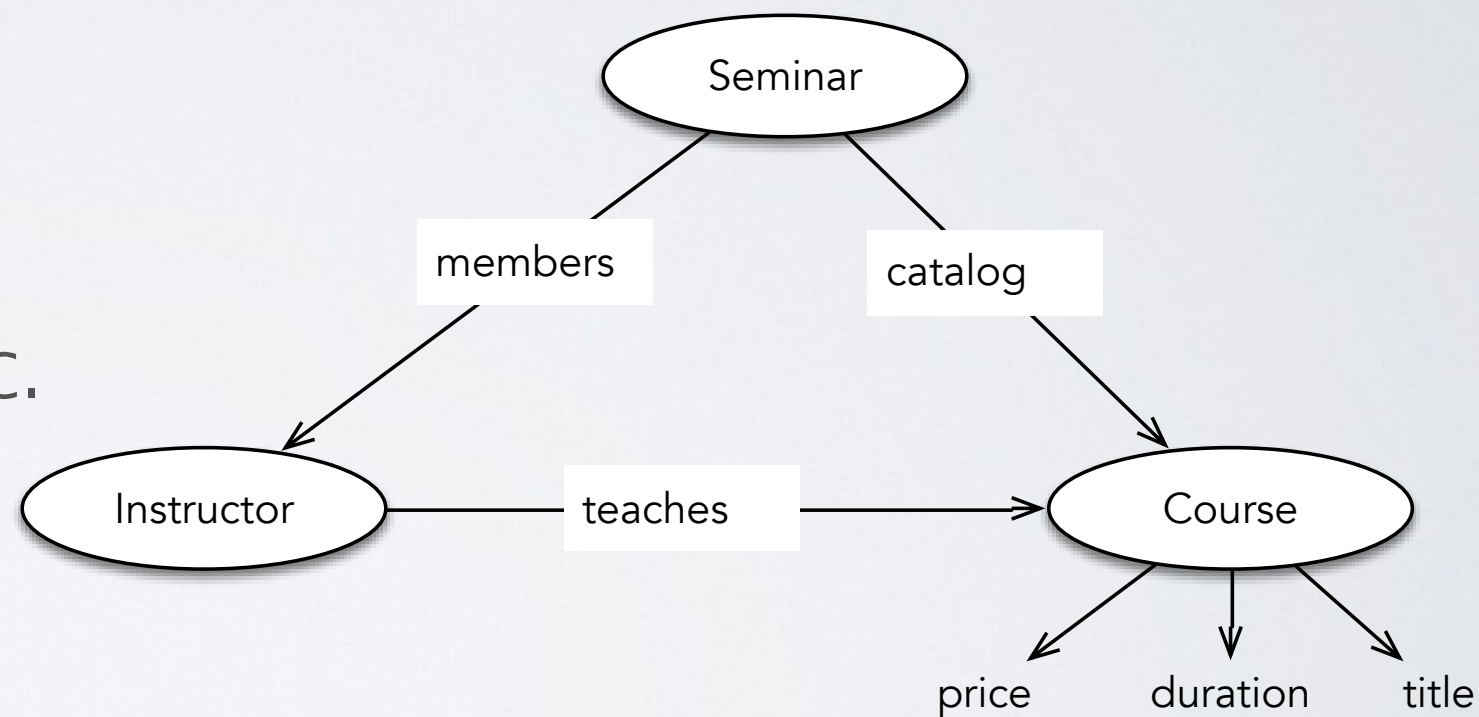
Column Name	Short Description	Meaning
EventId	Unique Identification for Each Event	Each event is assigned a unique 14-character alphanumeric code in the database. This code, used in conjunction with other primary keys (if applicable), are used to reference all database records. All database queries using a relational database (e.g., MS Access) should link tables using the ev_id variable.
InvestigationType	Type of Event	Refers to a regulatory definition of the event severity. The severity of a general aviation accident or incident is classified as the combination of the highest level of injury sustained by the personnel involved (that is, fatal, serious, minor, or none) and level of damage to the aircraft involved (that is, destroyed, substantial, minor, or none). The
AccidentNumber	NTSB Number	Each accident/incident is assigned a unique case number by the NTSB. This number is used as a reference in all documents referring to the event. The first 3 characters are a letter abbreviation of the NTSB office that filed the report. The next 2 numbers represent the fiscal year in which the accident occurred. The next two letters indicate the investigation category (Major, Limited, etc) and mode (Aviation, Marine, etc). The next three digits indicate the chronological sequence in which the case was created within the given fiscal year. And a final letter (A, B, C, etc) may exist if the event involved multiple aircraft
EventDate	Event Date	The date of the event. Dates are be entered in the format: MM/DD/YYYY
Location	Event Location Nearest City	The city or place location closest to the site of the event.
Country	Event Country	The country in which the event took place.
Latitude	Event Location Latitude	Latitude and longitude are entered for the event site in degrees and decimal degrees. If the event occurred on an airport, the published coordinates for that airport can be entered. If the event was not on an airport, position coordinates may be obtained using Global Positioning System equipment or nearest known reading.

# Concept Map [Novak 84]

- A mind map is a graphical representation of a declarative knowledge base with a hierarchical organization.
- Informal but structured representation of related terms in the domain.

# Concept Map (2)

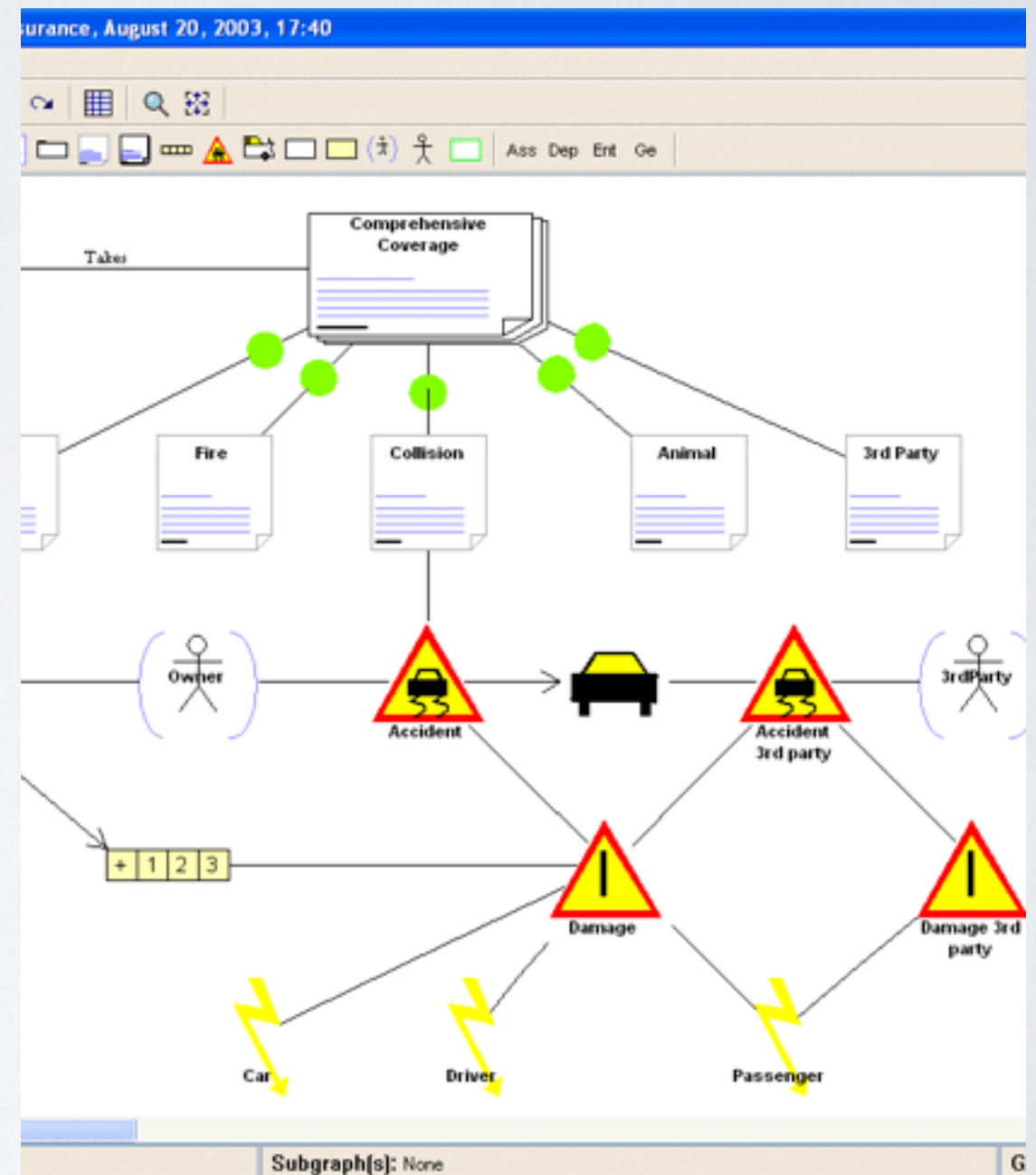
- Selection of pertinent concept.
- Hierarchical organization, from the generic to specific.
- Links between concepts: casualty, consequence, inclusion).





# Domain Specific Languages

- Computer language specialized to a particular application domain.
- Graphical or textual.



# Effective Use Cases

# Use Cases

- A Use Case is a particular way of using the system.
  - A sequence of interactions between the system and one or more actors.
- They are represented by UML Interaction diagrams (also called scenarios in domain analysis).
- A Use Case is a text: writing good Use Cases is a question of *style*.

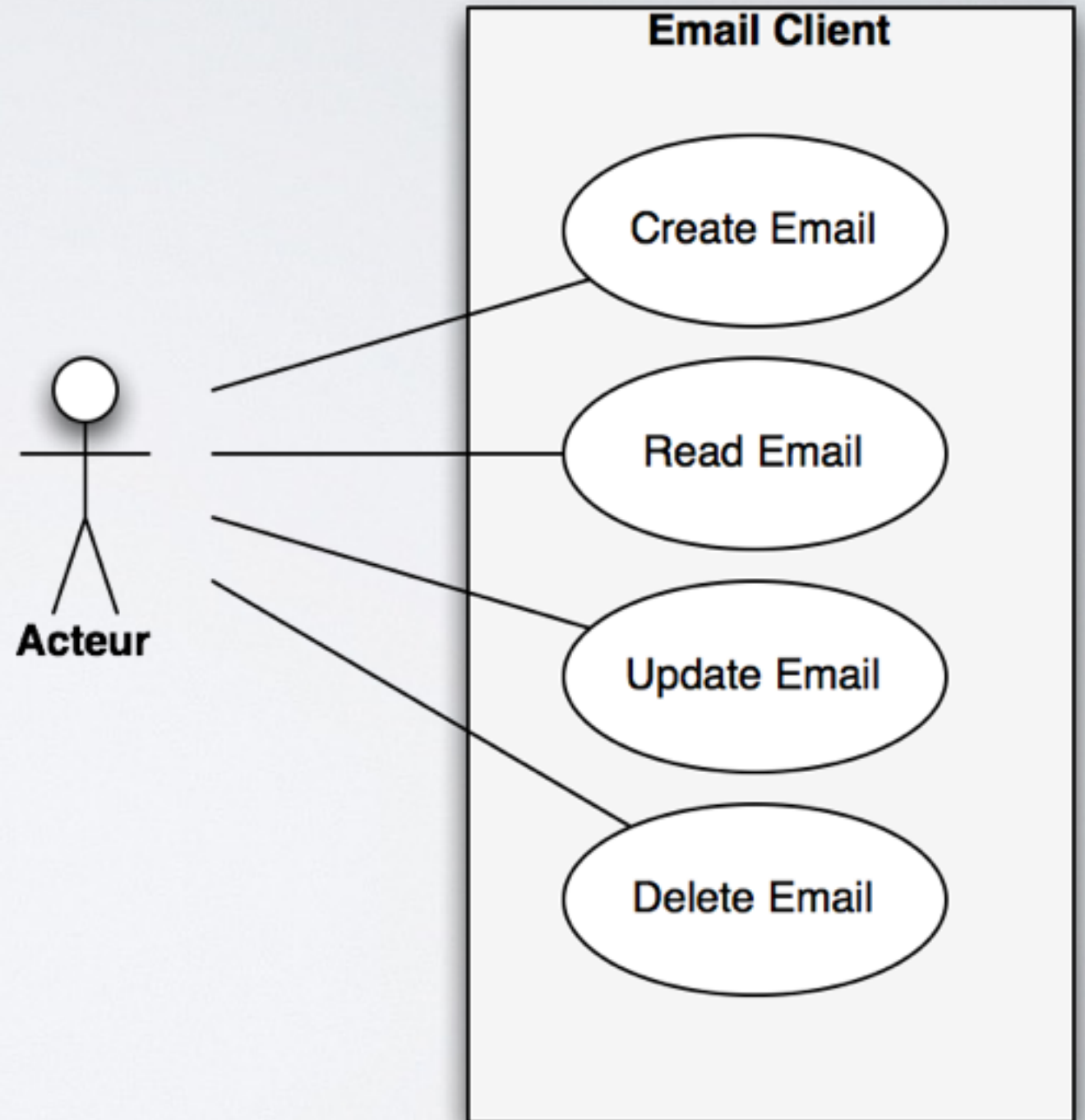
# The System as a Set of Services

- The compilation of all Use Cases describes informally the services provided by the System.
- They provide a description of the functional requirements.
- They can drive the progression of an iterative development model.



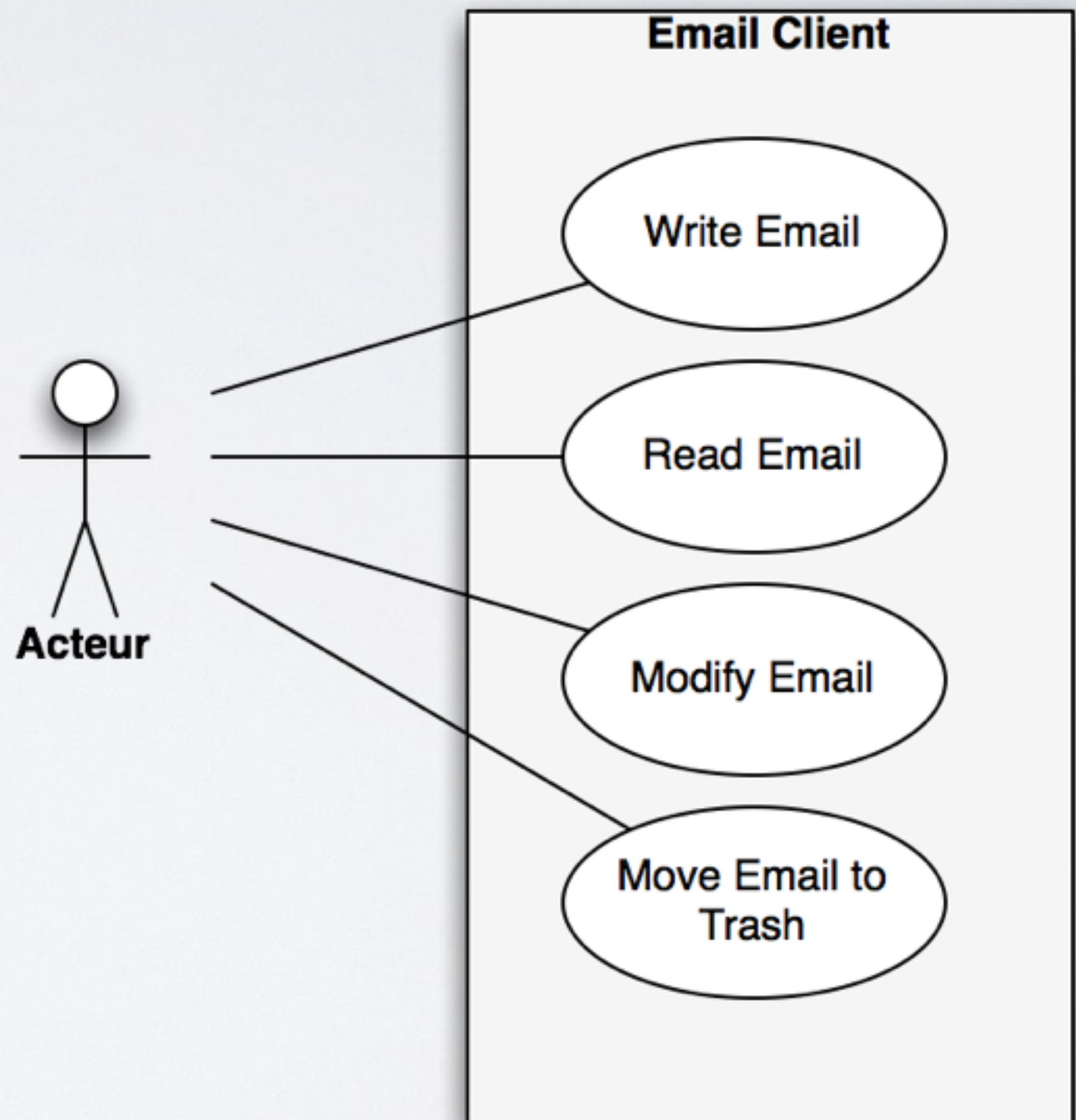
# Identifying Use Cases

- Use Cases are complex actions.
- They are described by several scenarios.
- Typically CRUDS are bad choices: simple actions, realized by the same actor to reach a single goal within the same security level.

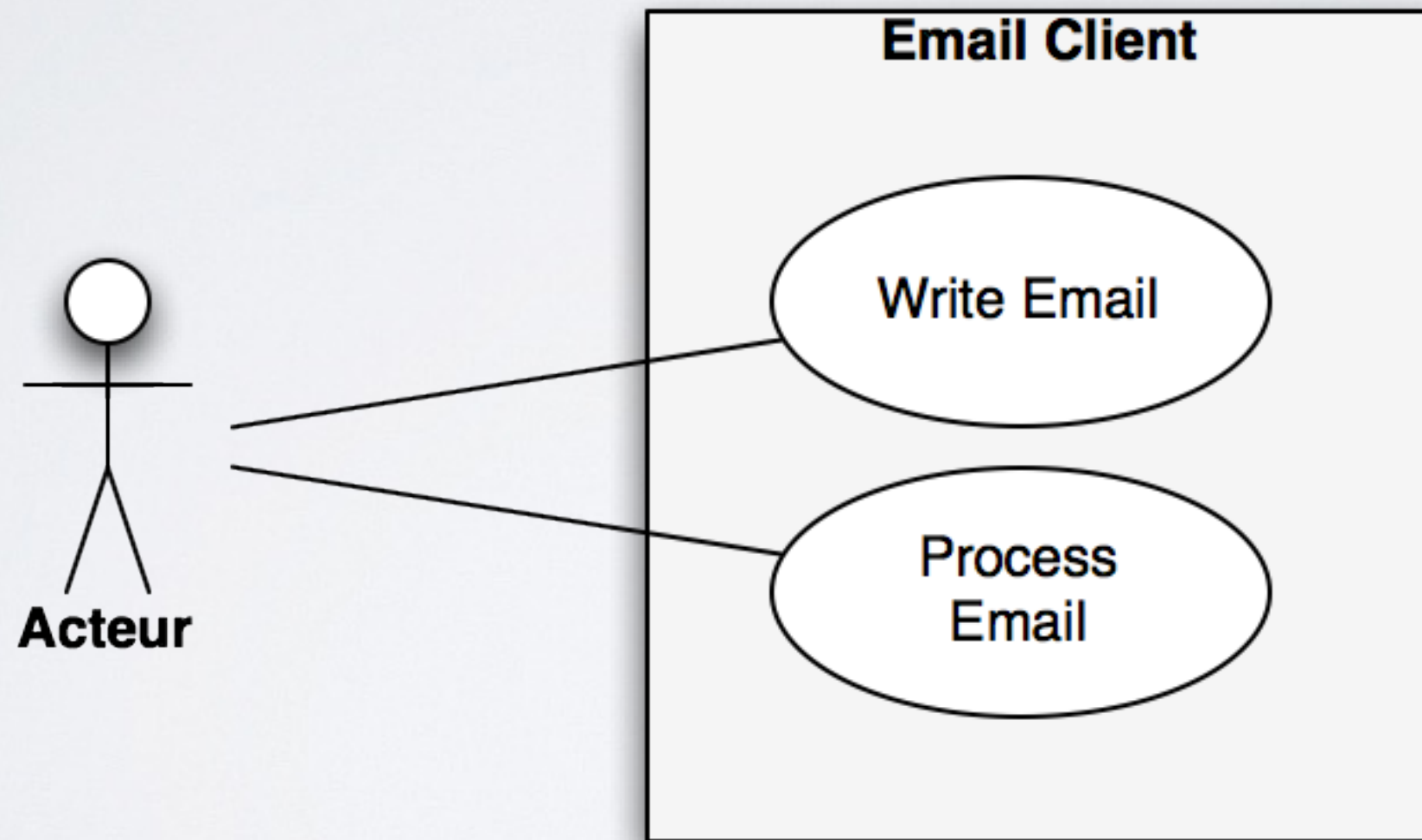


# Avoid CRUD Synonyms

- Differentiate Use Case from Action.
- An Use Case is a set of Actions.
- An action can be seen as an instance of an Use Case (Class/Object analogy).



# Use Cases should be named using the domain terminology.



# Informal Descriptions

- Use Cases are essentially text.
- They can follow different templates:
  - Rational Unified Process.
  - Cockburn.



# Use Case Template (Cockburn)

Use Case	# - the name should be the goal as a short active verb phrase
Goal in Context	a longer statement of the goal, if needed
Scope	what system is being considered black-box under design
Level	one of: Summary, Primary task, Subfunction
Success End Condition	
Failed End Condition	
Primary Actor	a role name for the primary actor, or description
Trigger	
Priority	how critical to the system / organization
Frequency	how often it is expected to happen
Pre-conditions	
Post-conditions	

Main success scenario	1. Description of Initial Action 2. Description of Action 2 3. (...) 4. Description of Last Action
Extensions	<#> : <condition> : <action or use-case> <#> : <condition> : <action or use-case>
Variations	<#> : <action or use-case>
Superordinate Use Case	optional, name of use case that includes this one
Subordinate Use Cases	optional, depending on tools, links to sub.use cases
Performance Target	the amount of time this use case should take
Open Issues	
Schedule	
Constraints	
Annexes	

# Template Example

Use Case	5 Buy Goods
Goal in Context	Buyer issues request directly to our company, expects goods
Scope	Company
Level	Summary
Success End Condition	Buyer has goods, we have money for the goods.
Failed End Condition	We have not sent the goods, Buyer has not spent the money.
Primary Actor	Buyer, any agent (or computer) acting for the customer.
Trigger	Purchase request comes in.
Priority	top
Frequency	200/day
Pre-conditions	We know Buyer, their address, etc.
Post-conditions	

Main Success Scenario	<ol style="list-style-type: none"> <li>1. Buyer calls in with a purchase request.</li> <li>2. Company captures buyer's name, address, requested goods, etc.</li> <li>3. Company gives buyer information on goods, prices, delivery dates, etc.</li> <li>4. Buyer signs for order.</li> <li>5. Company creates order, ships order to buyer.</li> <li>6. Company ships invoice to buyer.</li> <li>7. Buyers pays invoice.</li> </ol>
Extensions	<ol style="list-style-type: none"> <li>3a. Company is out of one of the ordered items: <ol style="list-style-type: none"> <li>3a1. Renegotiate order.</li> </ol> </li> <li>4a. Buyer pays directly with credit card: <ol style="list-style-type: none"> <li>4a1. Take payment by credit card (use case 44)</li> </ol> </li> <li>7a. Buyer returns goods: <ol style="list-style-type: none"> <li>7a. Handle returned goods (use case 105)</li> </ol> </li> </ol>
Variations	<ol style="list-style-type: none"> <li>1. Buyer may use phone in, fax in, use web order form, electronic interchange</li> <li>7. Buyer may pay by cash or money order check</li> </ol>



Superordinate Use Case	Manage customer relationship (use case 2)
Subordinate Use Cases	<ul style="list-style-type: none"> <li>• Create order (use case 15)</li> <li>• Take payment by credit card (use case 44)</li> </ul>
Performance Target	5 minutes for order, 45 days until paid.
Channel to primary actor	May be phone, file or interactive.
Secondary Actors	Credit card company, bank, shipping service
Open Issues	<ul style="list-style-type: none"> <li>• What happens if we have part of the order?</li> <li>• What happens if credit card is stolen?</li> </ul>
Schedule	Due Date: release 1.0
Constraints	
Annexes	

# Final Remarks

# Use Cases and User Interface Modeling

- Use Cases and User Interfaces (UI) should be different, independent things.
- Use storyboards (or other techniques) to model UI.
- Avoid UI related actions. For instance: “The user chooses in the List Box and clicks on the Validate button.

# Iterate

- Good Use Cases need several iterations.
  - It is virtually impossible to write a complete Use Case at once.
  - Use first versions as a base for discussions and improvements. Compare models.
  - Use Cases and Conceptual Classes should evolve together.
- Stop before reaching perfection.
  - Aim quality targets: costs, deadlines, etc.



# Use UML Diagrams

- Describe Actions using:
  - Object diagrams (snapshots).
  - Sequence diagrams.
- Describe Use Cases using Activity Diagrams.
- Describe different states of a conceptual class using State Machines.
- Do not try to exploit all UML modeling possibilities.

# Cockburn's Writing Reminders

1. Start from the top and create a coherent story line.
2. Name the use cases with short verb phrases that announce the goal to be achieved.
3. Focus on the main scenario.
4. Be concise yet pertinent (avoid long documents).
5. Write full sentences with active verb phrases, in the present tense.

6. Make sure the actor is visible in each step.
7. Put alternative behaviors in the extensions, rather than in if statements in the main body.
8. Identify the correct goal.
9. Include sub use cases, use UML «include» relationships to represent them.
10. Start from the trigger, continue until the goal is delivered or abandoned.
11. Keep the GUI out.

# Conclusion



# Conclusion

- «Things must be as simple as possible, but no simpler». [A. Einstein]
- «Un bon modèle n'est pas un modèle où l'on ne peut plus rien ajouter, mais un modèle où on ne peut plus rien enlever.» [A. de St-Exupéry]
- An incomplete model is not a simple model.

# References

- Objects, components, and frameworks with UML: the Catalysis Approach, by Desmond D'Souza and Alan Wills. Addison Wesley, 1998.
- <http://www.drdobbs.com/how-to-avoid-use-case-pitfalls/184414560>
- Writing Effective Use Cases. Alistair Cockburn.
- UML Distilled: A Brief Guide to the Standard Object Modeling Language. Martin Fowler.
- Fowler, M., (1997) Analysis Patterns – Reusable Object Models, Indianapolis: Addison Wesley.
- Domain Modeling. Harsh Jegadeesan's Classroom. BITS Pilani.