Avoiding OCL specification pitfalls

Teaching Software Modeling by Using Constraints

Dan CHIOREAN*, Ileana OBER** and Vladiela PETRASCU*
*Babes-Bolyai University of Cluj - **UPS Toulouse & IRIT

Educators Symposium @ MODELS 2011, October 18th 2011

Motivation

- The success rates for software projects: ≈60% in case of iterative & agile processes and $< 50\%$ in other cases
http://drdobbs.com/architecture-and-design/226500046?pgno=2

- MDE asks for a **clear** and **complete** model specification

- Design by Contract is a technique for producing **rigorous models**

- The deployment of Design by Contract is still limited - we believe this should change

Remarks on current OCL training approaches

- **OCL training is:**
  - **getting more widespread**, although less than other modeling techniques
  - **focused on technical aspects concerning the constraint language**

- **great available resources** - books, articles and slides on OCL : Warmer, Gogolla, Hussmann, Demuth, Atkinson, etc.

- still, **too many hasty OCL examples** (including in the standard)
  - *uncompilable*
  - not (completely) *compliant* to specifications
  - not (enough) *tested*
  - results vary depending on the OCL tool used
  - in case of constraint failure *no information useful for diagnostic*

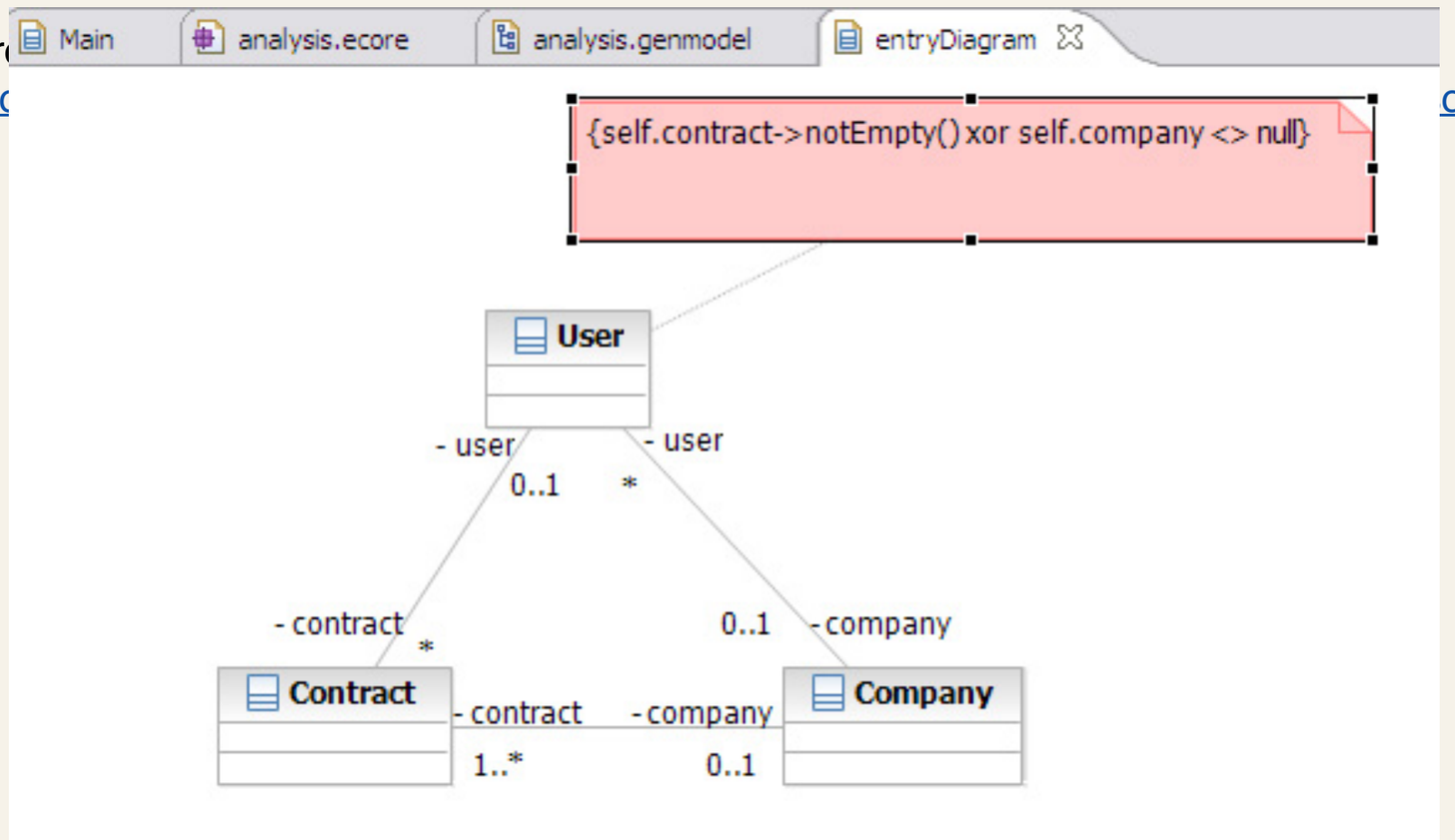
Our view on teaching modeling

- Highlights on the **complementarity of models and associated constraints**
- Argues for the **need of constraints, due to the value added** by means of **suggestive examples**
- Presents **principles & best practices** for specifying constraints
- **Stresses out the role** of **complete** and **unambiguous requirements**
- **Illustrates** the importance of the **rigor** in specifying constraints
- **Emphasizes the compulsoriness** of a full conformance between requirements & specifications

Understanding the problem and the problem domain

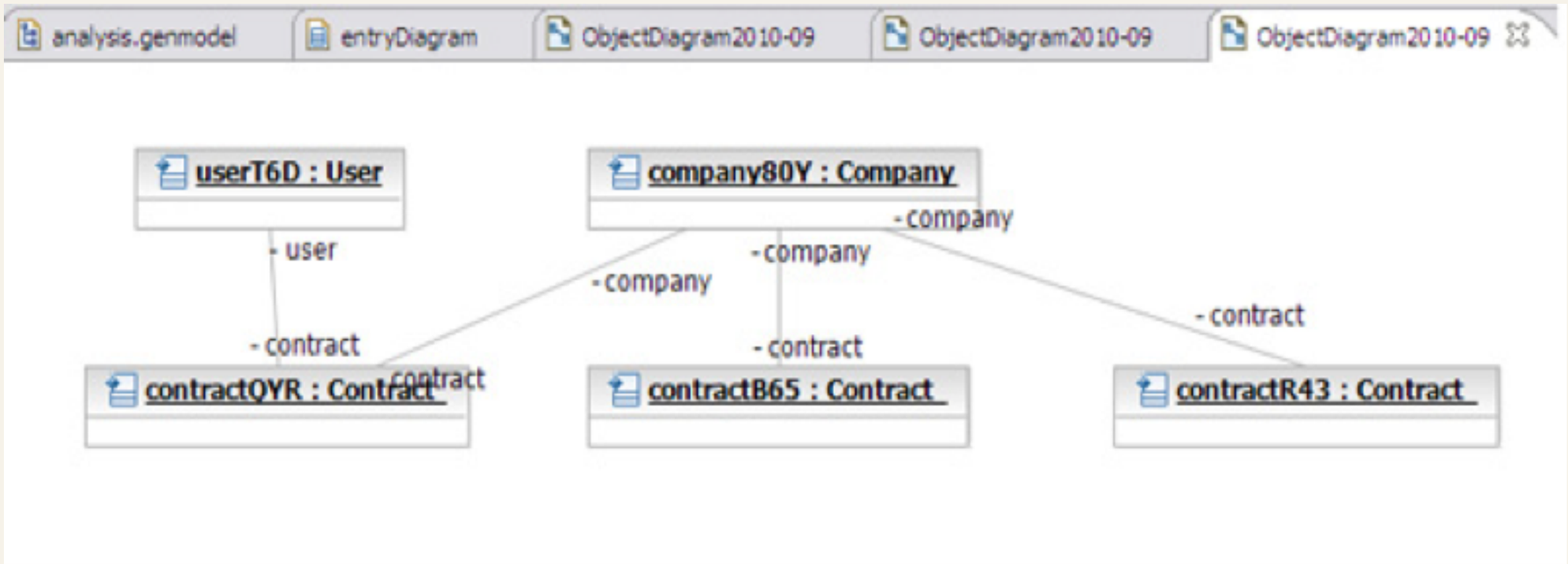
"… the library offers a subscription to each person employed in an associated company. In this case, the employee does not have a contract with the library but with the society he works for, instead."

A. Todorova - Produce mor https://www.ibm.com/develc oftware-architect/



Understanding the problem and the problem domain

Using snapshots to clarify requirements

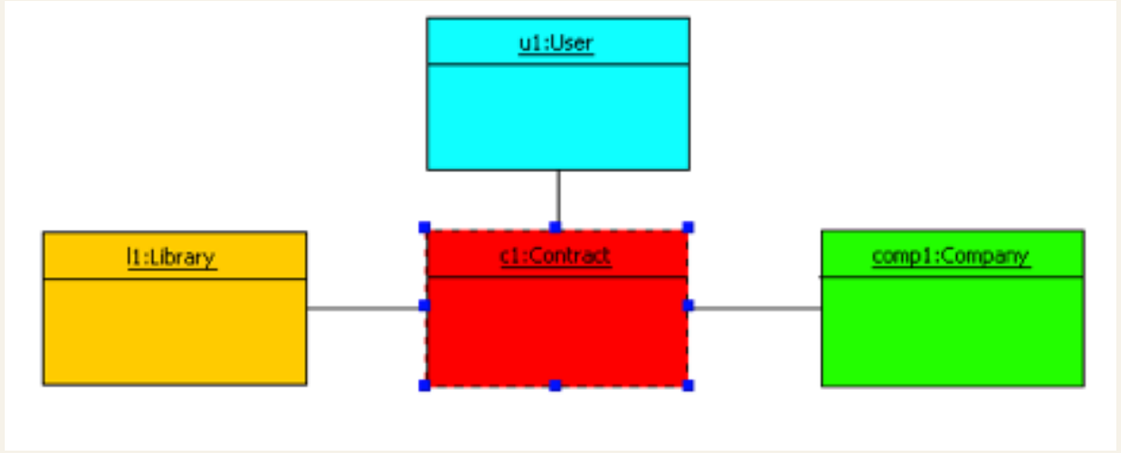


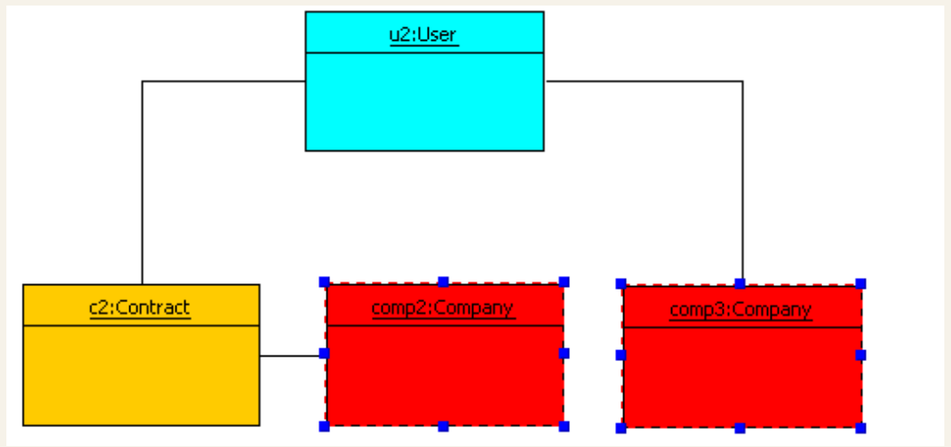
context User
  inv TodConstraint:
    self.contract->notEmpty() **xor self**.company <> null

Understanding the problem and the problem domain

Using snapshots to test constraints

Use suggestive snapshots to clarify requirements & test constraints
Ex. Unwanted model instantiations that are not caught by the invariant proposed in [Tod11]



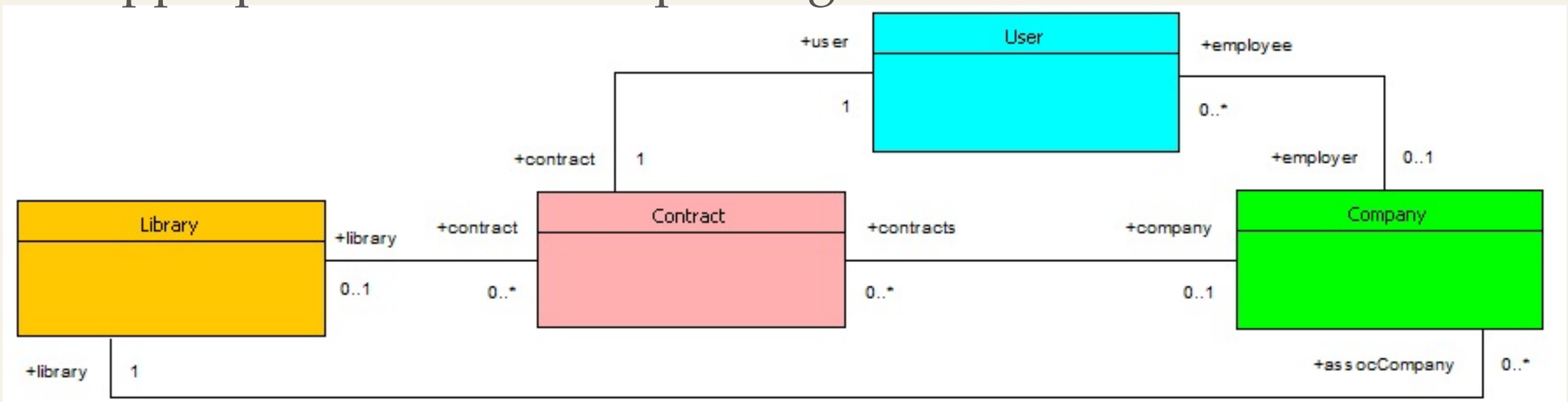


context User
inv TodConstraint:
self.contract->notEmpty() **xor self**.company <> null

Understanding the problem and the problem domain

Improving the model & associated constraints

- Improve the existent solution by updating the model, choosing an appropriate context & updating the constraints



context User
inv theContractIsWithTheEmployer:
if self.employer->isEmpty
then self.contract.library->notEmpty
else self.employer = self.contract.company
endif

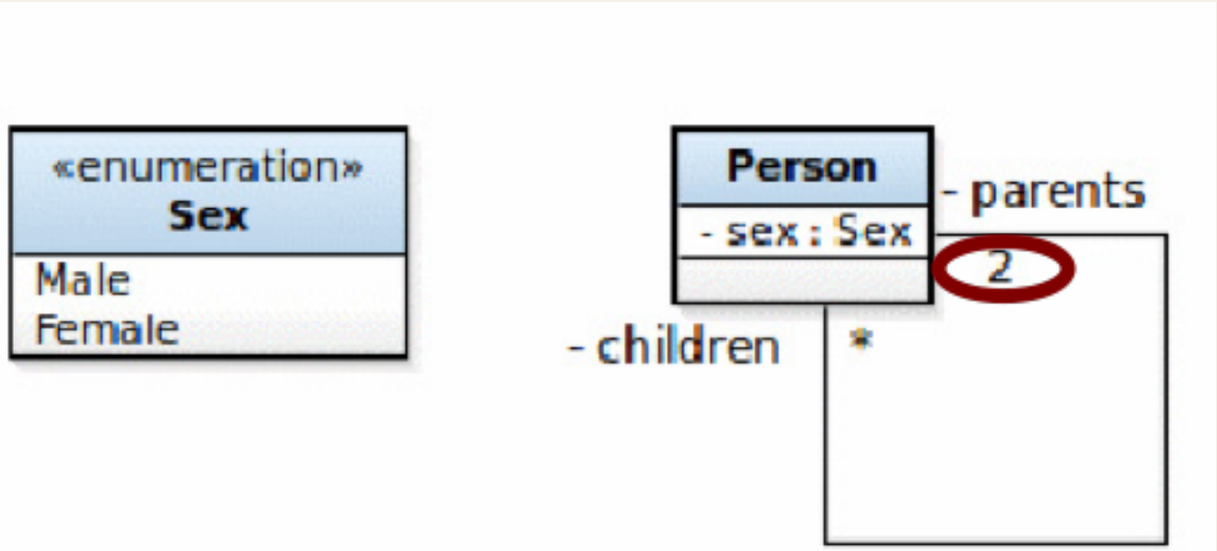
context Contract
inv onlyOneSecondParticipant:
self.library->isEmpty **xor** self.company->isEmpty

Specifying the intended usage of the model

- The ultimate role of models is to produce software

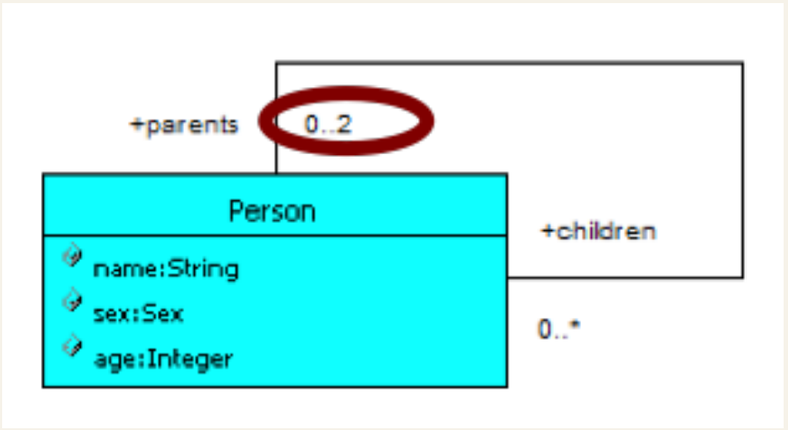
self.parents->asSequence()->at(1).sex <>
self.parents->asSequence()->at(2).sex

context Person
 inv parentsSex:
  let PS: Set(Sex) = self.parents.sex->asSet **in**
  PS->size = 2 **and** PS->excludes(Undefined)



Thinking rigorously about the data validity

context Person
  inv notSelfParent:
    self.parents->*select*(p | p = **self**)->*isEmpty*
context Person
  inv parentsAge:
    self.parents->*reject*(p | p.age - **self**.age >= 16)->*isEmpty*
context Person::addChildren(p:Person)
  pre childrenAge:
    self.children->*excludes*(p) **and self**.age - p.age >= 16
  post chidrenAge:
    self.children->*includes*(p)



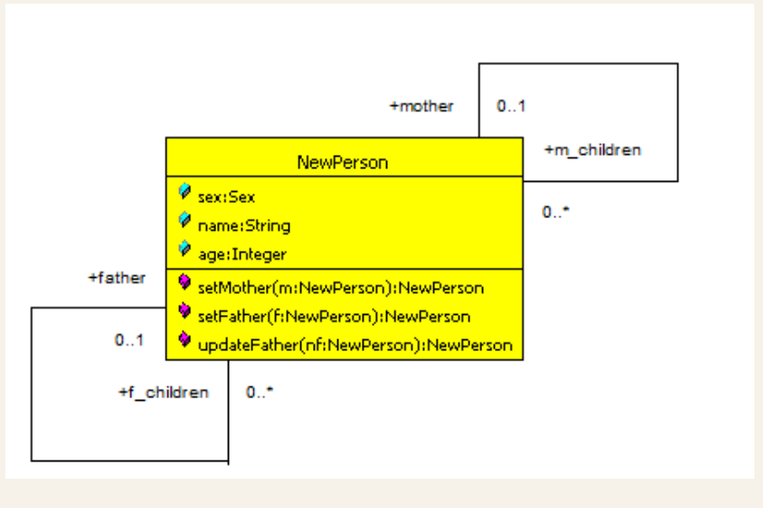
Using finer constraints if possible

context Person
inv parentsSexN:
let f: Person = **self**.parents->at(1) **in**
let m: Person = **self**.parents->at(2) **in**
not f.oclIsUndefined() **implies** f.sex = Sex::female **and**
not m.oclIsUndefined() **implies** m.sex = Sex:: male

Choosing from different model proposals

context NewPerson
inv parentsSex:
(**self**.mother->*size* = *1*
implies self.mother.sex = Sex::female) **and**
(**self**.father->*size* = *1* **implies**
self.father.sex = Sex::male)

context NewPerson
inv parentsAge:
((**self**.mother->*size* = 1) **implies**
(**self**.mother.age - **self**.age >= 16))
and ((**self**.father ->*size* = 1) **implies**
(**self**.father.age - **self**.age >= 16))



Sum-up

- Complementing the model specification with constraint specification is essential and opens the way to:
  - ✦ deeper problem understanding
  - ✦ rigorous model description
  - ✦ unwanted model instances detection
- Conformance with requirements - first quality criterion in constraint specification
- Abstraction - main principle that should drive the modeling
- Identifying and specifying constraints follow abstraction

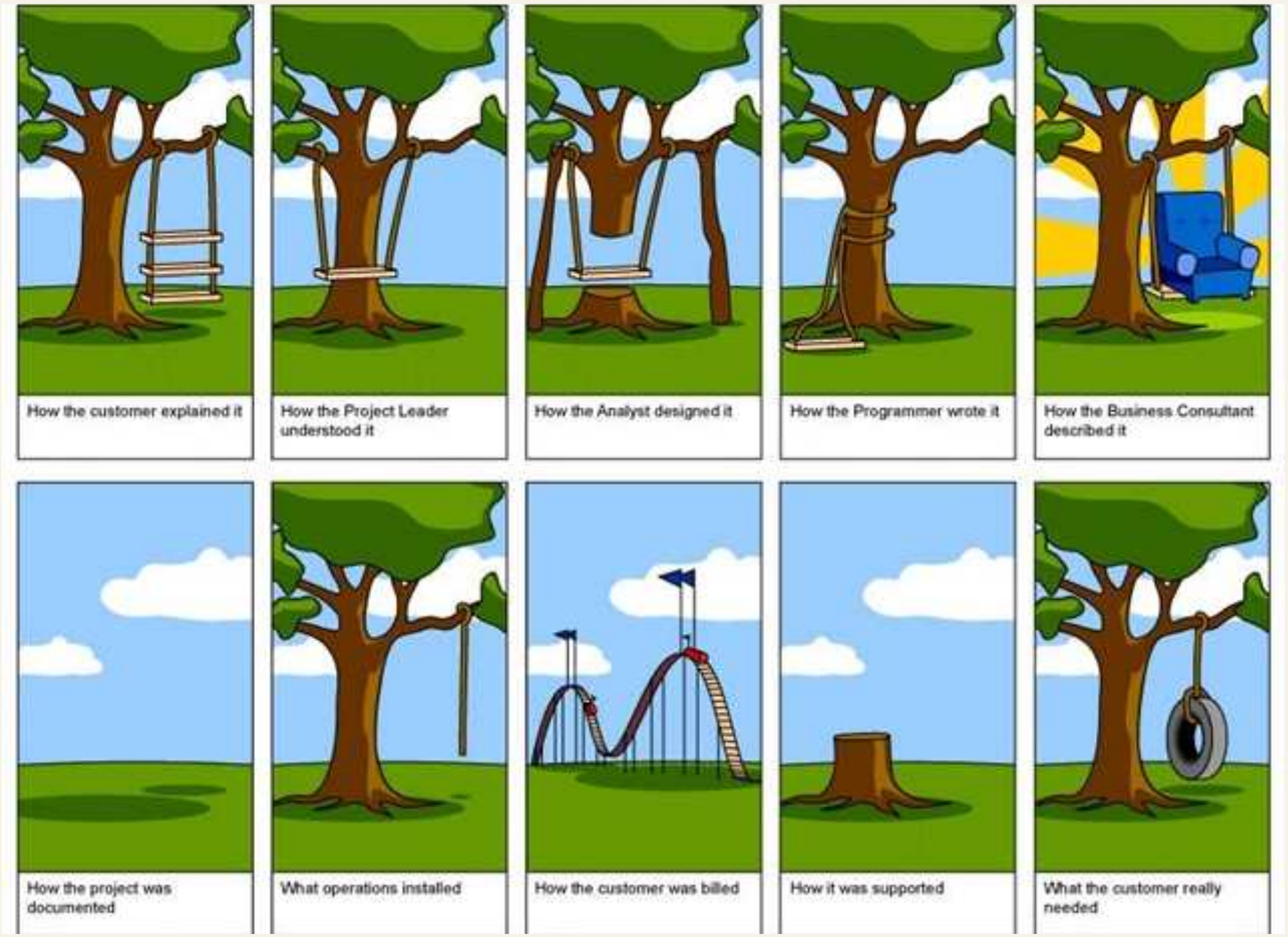
Conclusions (1/2)

- **Implementation of the proposed approach** - (under)graduate modeling courses at Babes-Bolyai University of Cluj-Napoca & UPS Toulouse

- **Encouraging** results:
  - **much easier to convince students** with respect to the value of Design by Contract,
  - **higher quality of resulting OCL specifications**,
  - **students able to evaluate the quality of OCL specifications found on the web and to choose high quality examples**
- **Modeling and programming share much more similarities than**

Conclusions (2/2)

- **Rigor** - the main attitude in specifying constraints
- **Hastiness** - the main enemy in the constraint specification process
- **The intended usage of the system** - has to be captured by requirements
- **Both constraint and model specification** should be done **-** in an **incremental and iterative process**
- **Teaching / training** - should realistically illustrate the positive influence of (OCL) constraints on model specification
- **Pragmatism** - should be the leading attitude in teaching (examples, specification tips and tricks, specification patterns)



When model specification process is hasty...

Date