Data and Process Modelling

2. Information System Development

Marco Montali

KRDB Research Centre for Knowledge and Data
Faculty of Computer Science
Free University of Bozen-Bolzano

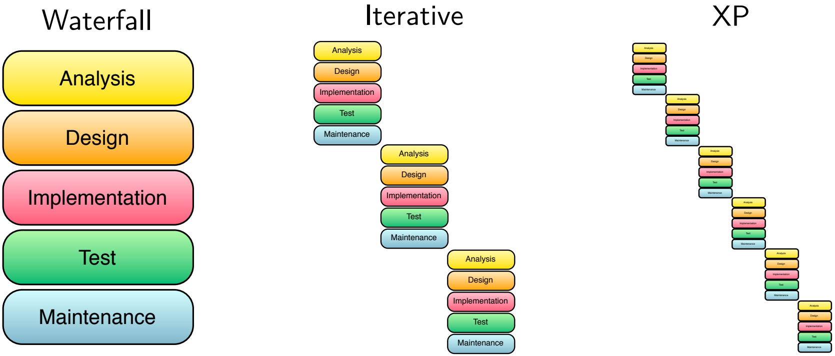
A.Y. 2014/2015

IS Engineering

Development of an IS: problem-solving.

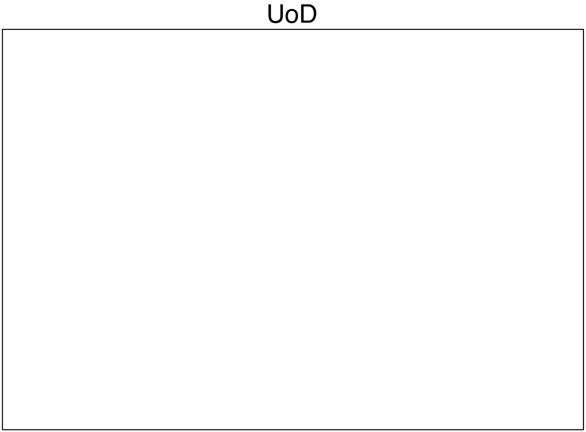
1. **Analysis** (includes conceptual modeling): what the IS is about, what are the requirements.
2. Design (includes logical modeling): how to accomplish the requirements.
3. Implementation: coding of the design under specific architectural/technological choices.
4. Testing: check if the implementation works and meets the requirements.
5. Maintenance: assist users after release, keep the IS working.

IS Engineering Processes



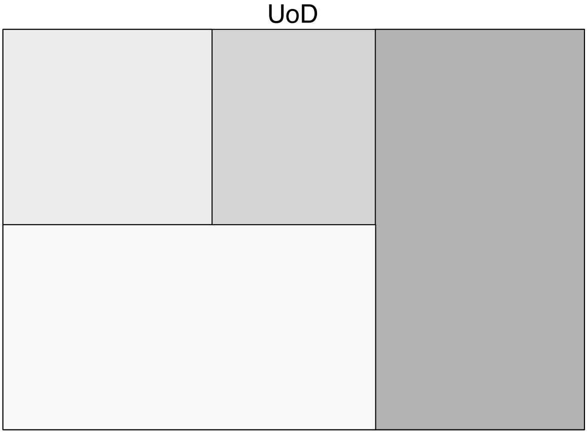
Incremental Iterative Approach

Divide et impera.



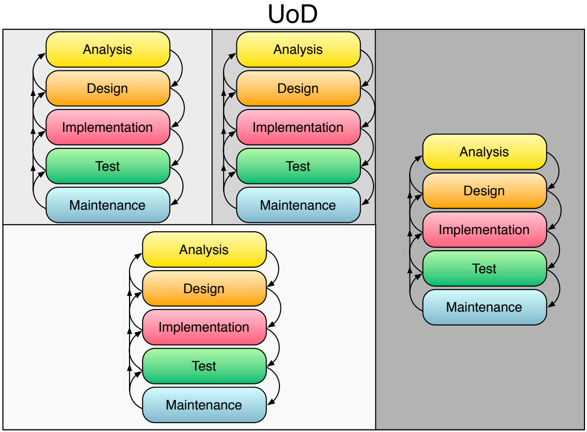
Incremental Iterative Approach

Divide et impera.



Incremental Iterative Approach

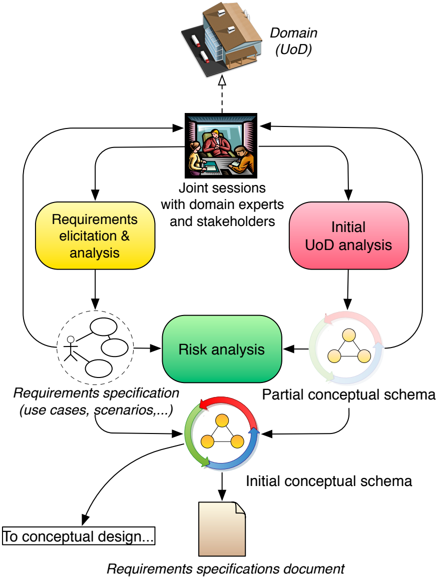
Divide et impera.



Refined Steps

1. Feasibility study: is the idea implementable?
2. **Requirements analysis**: what should the system do?
3. **Conceptual design - data, processes**: what is the conceptual schema modeling the UoD?
4. **Logical design - data, processes**: how can the conceptual schema be translated into a logical schema?
5. Basic physical design - data and processes: how can the logical schema be represented in a concrete management system?
6. Basic external design - data and processes: which information can be accessed by users, and how?
7. Prototyping: how does the IS look like?
8. Completion of design.
9. Implementation of production version.
10. Testing and validation: does the IS work well and satisfy the requirements?
11. Release of software, documentation, training.
12. Maintenance.

Requirements analysis

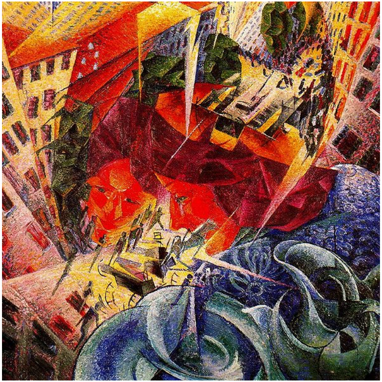


First delineation of the **IS to be**.

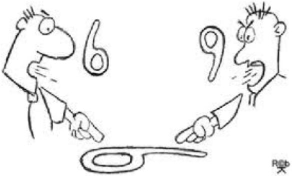
- Relevant documentation examined.
- Meetings with domain experts, intended users, policy makers, stakeholders.
- Prioritization of the next steps.
- Output: requirements specifications document that clearly describes functional/non-functional requirements, and sketches an initial conceptual schema of the IS to be.
  - Contract!

Interaction with Domain Experts

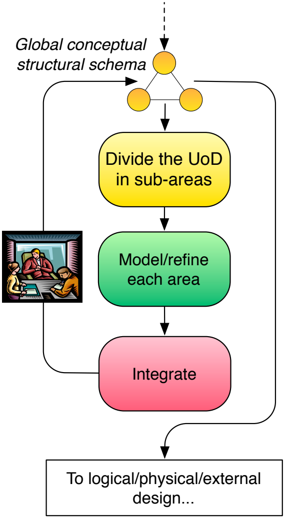
M. C. Escher - Up and down



Umberto Boccioni - Visioni simultanee



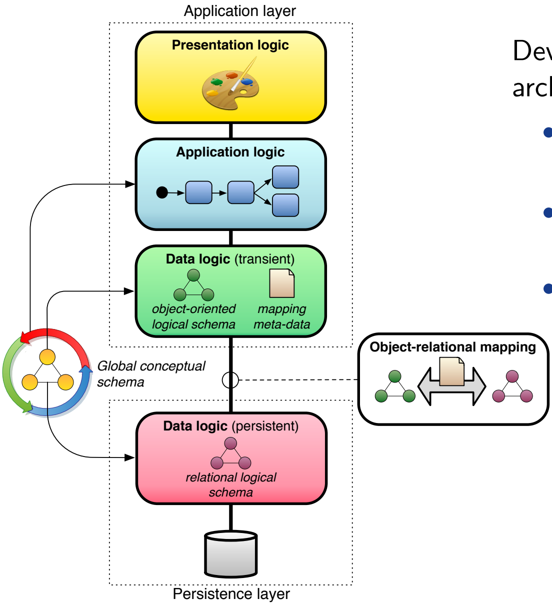
Structural Conceptual Modeling - Phases



Development of the structural conceptual schema.

- Continuous involvement of domain experts.
  - Definition of a glossary of terms.
- Incremental iterative approach.
  - Start from the initial structural conceptual schema.
  - Iterate...
    1. Split UoD into (overlapping) sub-areas, with priority.
    2. Generate/refine the conceptual schema of each area.
    3. Integrate the sub-schemas into a global conceptual schema.

Logical/Physical Design without Explicit Processes



Development of a typical 4-tier architecture.

- Mirrors in the physical design.
- Processes embedded in the application logic.
- Two similar logical information schemas, two similar physical databases:
  - transient - data logic of the application (typically: OO).
  - persistent - data logic of the persistence layer (typically: relational).

Zachman's Framework

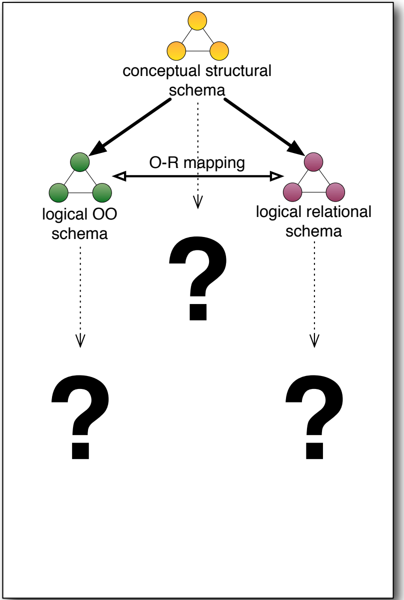
Partitioning of the IS abstract architecture.

- Vertical partitioning: levels of abstraction.
- Horizontal partitioning: aspects/concerns.

	Why (Motivation)	**What (Data)**	**How (Function)**	**When (Time)**	**Who (People)**	**Where (Network)**
Contextual	goal list	material list	process list	event list	organizational unit&role list	geographical locations list
Conceptual	goal relationship	structural conc. model	process model	event model	organizational unit&role model	locations model
Logical	rules diagram	data model diagram	process diagram	event diagram	role relationship diagram	locations diagram
Physical	rules specification	data entity specification	process function spec.	role specification	location specification	event specification
Detailed	rules details	data details	process details	event details	role details	location details

- Logical and physical layers can be split into transient and persistent.
- Mappings between levels to be considered.

Which Languages???



Conceptual Modeling Languages: Criteria

Expressibility: the measure of what can be modeled.

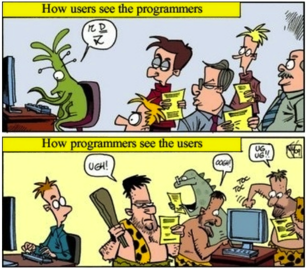
100% principle

The language should be able to express all the *relevant* static and dynamic aspects of the UoD.

- Remember: the conceptual schema is the knowledge component of the IS.
- 100% principle $\rightarrow$ completeness: the conceptual schema captures all the required knowledge.
- Completeness $\rightarrow$ quality.
- Correctness:
  - Syntactic: conformance to the language meta-model.
  - Semantic: knowledge of conceptual schema is relevant and *true* in the domain.
- Completeness is a goal, correctness is a requirement!

Conceptual Modeling Languages: Criteria

Clarity: how easy the language can be understood and used (by different stakeholders).



- Graphical vs textual notations.
- The language must be unambiguous: formal foundation.
- The more expressive the language, the more difficult is to retain clarity.
- Less expressive languages require complex combinations of their few constructs.

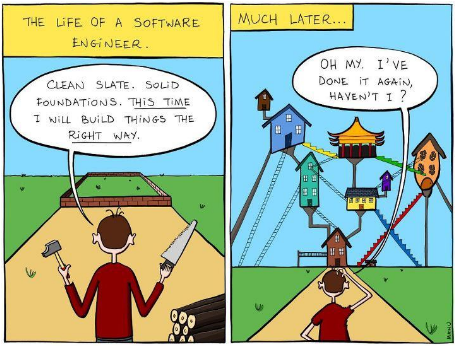
- Abstraction: remove unnecessary details. Use requirements to drive abstraction.
- Simplicity: Prefer simple schemas. Follow *Occam's razor* with a critical approach.
- Orthogonality: minimization of the overlapping of language constructs. Their (in)dependence must reflect the one of the corresponding domain aspects.

Conceptual Modeling Languages: Criteria

Semantic relevance: modeling of conceptually relevant aspects only.

Conceptualization principle

A conceptual model should only include conceptually relevant aspects of the UoD, excluding all aspects of external/internal data representation, physical data organization and access as well as aspects of particular external user representation such as message formats, data structures, etc.



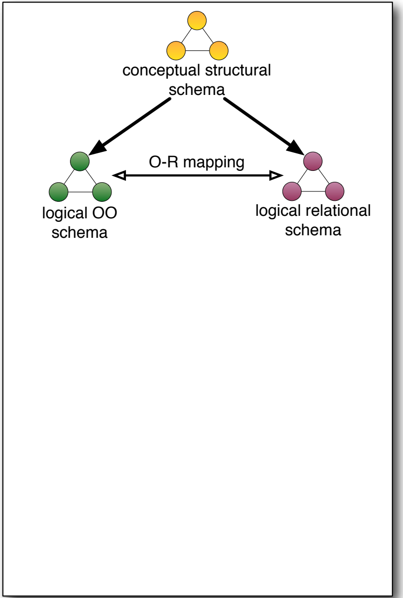
- Again, simplicity!
- Semantic stability: how well the model retains its original intent in the face of domain or requirements changes.
- Design-independence.
    - Design aspects are tackled during the design phase!
    - No architectural/design patterns.

Trade-Offs

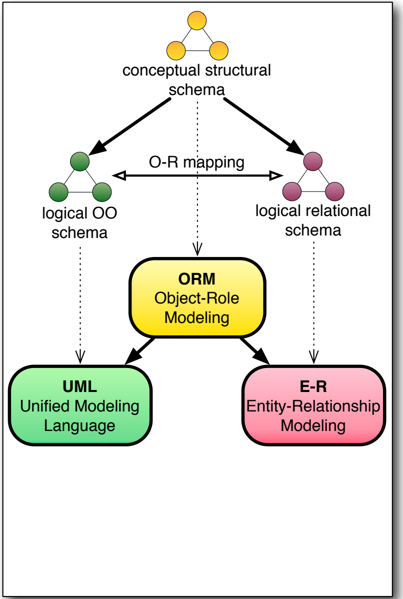
Trade-offs between contrasting desiderata.

- Expressivity vs tractability: the more expressive the language, the harder it is to *compute* with it.
- Parsimony vs convenience: fewer concepts vs compact models.
    - Would you use `Assembler` to implement a web server?

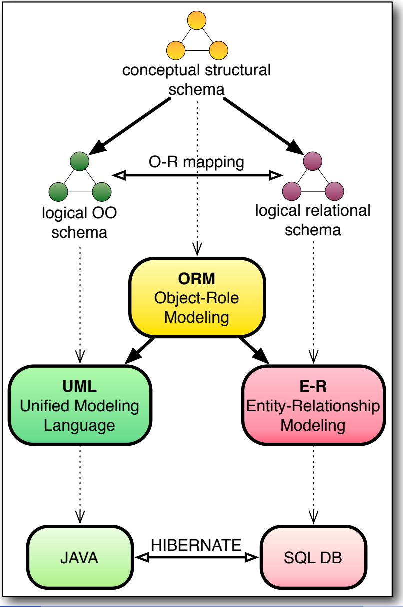
Modeling Languages and Frameworks



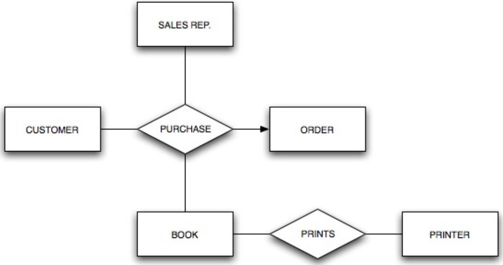
Modeling Languages and Frameworks

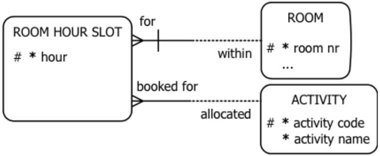


Modeling Languages and Frameworks



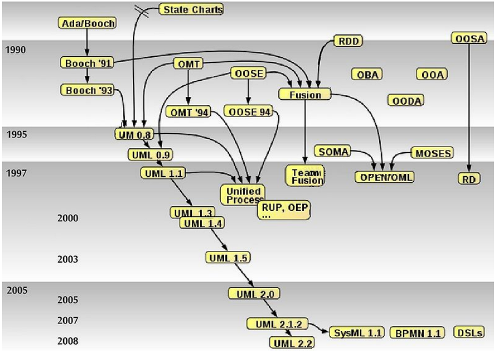
E-R: Abstract Representation of Data





- Introduced by Peter Chen (1976).
- The most widely used approach to data modeling.
- Key notions:
  - entities, relationships, attributes;
  - identification and multiplicity constraints.
- Independent from the target software platform.
- Lack of dynamic modeling.
- Close to relational database schemas $\rightarrow$ logical relational modeling!
- Different dialects: Chen, Barker, IE, IDEF1X, EXPRESS . . .

UML: Modeling Standard for OO Software Engineering

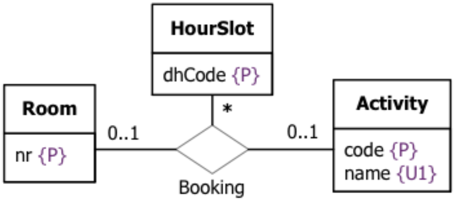


- Born from:
  - 3 amigos:
    - ★ Rumbaugh's Object-modeling technique;
    - ★ Booch's OO design;
    - ★ Jacobson's OO software engineering method.
  - Harel's state-charts.
- OMG standard since 1997.

- Family of notations:
  - Structure diagrams: class/object diagram, component, composite structure, deployment, package, profile.
  - Dynamic diagrams:
    - ★ Behavior: use case, state machine, activity.
    - ★ Interaction: communication, interaction overview, sequence, timing.

UML Class/Object Diagrams

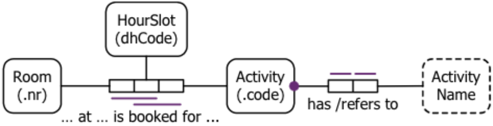
Structural modeling, especially for OO design → logical OO schemas.



- Behavioral aspects (operations/methods).
- Encapsulation policies (OO paradigm).
- Key notions:
  - object (class) as entity (type);
  - attributes (with visibility), relationships (basic, generalization, aggregation, composition);
  - multiplicity constraints, OCL;
  - behavioral aspects (operations, parameters, visibility);
  - no mandatory identification for objects (implicit reference, object ids).

ORM

Fact-oriented conceptual approach to modeling and querying the information semantics of a UoD.



- Introduced by Falkenberg in 1973; formalized and enhanced by Halpin (→ ORM 2).
- Starts from elementary facts.
- Key notions:
  - objects (relevant entities) playing roles (parts in relationship types);
  - intuitive treatment of n-ary (ordered) roles;
  - rich business constraints (subsumes UML and E-R);
  - no use of attributes!
- Diagrammatic + textual form (controlled natural language).
- Two forms of validation with domain experts:
  - verbalization - natural language description of the diagrams;
  - population - sample prototypical instances and counterexamples.

Absence of attributes in ORM

Claim: introducing attributes in the conceptual design is a premature commitment.

- In ORM fact structures are expressed as fact types (relationship types):
  - unary (e.g. Person *smokes*);
  - binary (e.g. Person *was born* on Date);
  - ternary (e.g. Person *visited* Country in Year);
  - ...
- Advantages:
  - semantic stability (minimize the impact of change caused by the need to record something about an attribute);
  - natural verbalization (all facts and rules may be easily verbalized in sentences understandable to the domain expert);
  - populatability (sample fact populations may be conveniently provided in fact tables);
  - null avoidance (no nulls occur in populations of base fact types, which must be elementary or existential).
- Attributes can be obtained through views over the ORM schema.