Software Engineering Approaches

Choosing the Approach

- *How to decompose the problem*
- *How to organize the system*

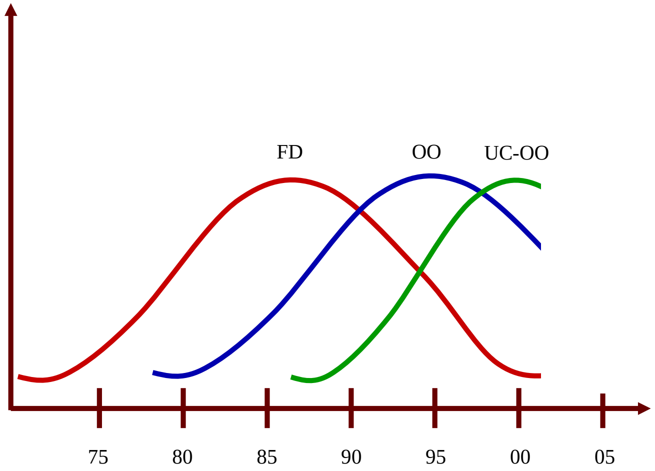
Drivers:

- Coping with size
  - Structured approach
  - Stepwise refinement
  - Hierarchical organisation
- Coping with change
  - Logic model
  - Maintainable results
- Coping with documentation
  - Simple notation
  - Graphical elements

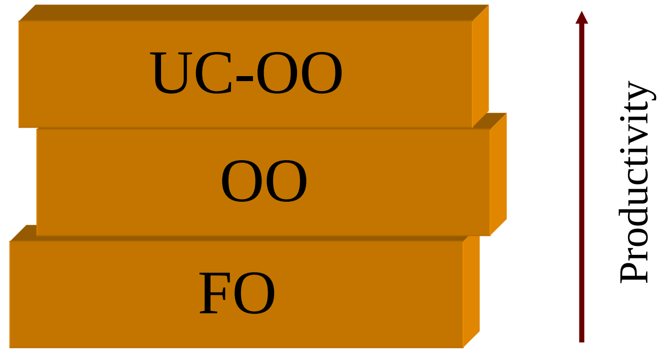
3 Important Paradigms

- Functional Orientation
  - Fast and straightforward development
  - Hard to maintain, short life time
  - Low reuse
- Object Orientation
  - Longer life time, easier to reuse
  - Requires high competence
  - High risk during development
- Use Case driven Object Orientation
  - Predictable development, low risk
  - Easy to maintain
  - Still not as fast as functional orientation

The Paradigms



Concepts, Abstractions, Principles, Patterns: **Adding** to the Language



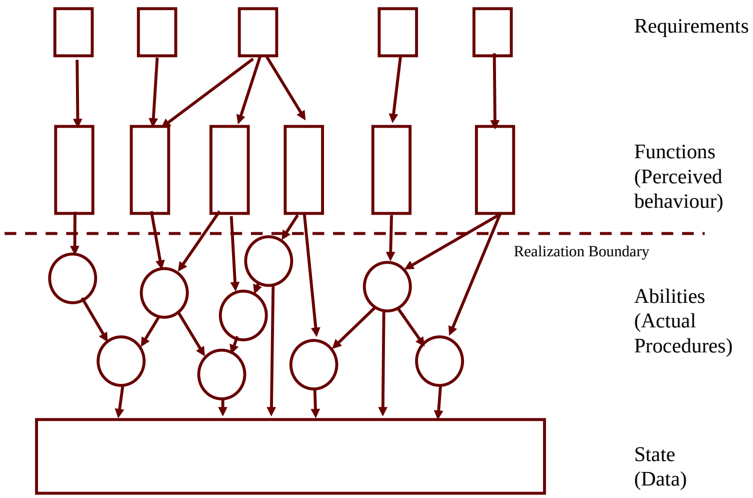
3 Important Paradigms

- Functional Orientation
- Object Orientation
- Use Case driven Object Orientation

Functional Orientation

- Focus on *function*: What does the system *do*?
- The system provides function by using its *abilities*.
- Abilities can be *decomposed* into finer grained abilities, to an arbitrary level.
- The *state* of the system *affects* the abilities but is a separate characteristic.
- Top-down functional decomposition.

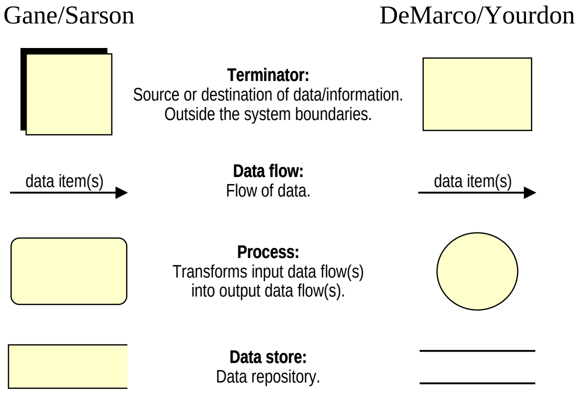
Functional organization



Structured Analysis (SA)

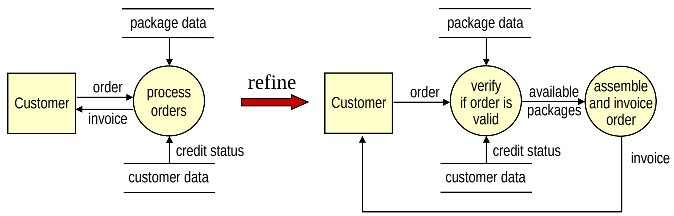
- Developed 1975/76
  - DeMarco/Yourdon
  - Gane/Sarson
- System = Process transforming input into output
- Hierarchical, logical system model
  - Processes
  - Data flows
  - Data stores
  - Terminators
- Notation:
  - Data flow diagrams (DFDs)
  - Data dictionary (DD)
  - Process specifications (PSpecs)

Data Flow Diagrams

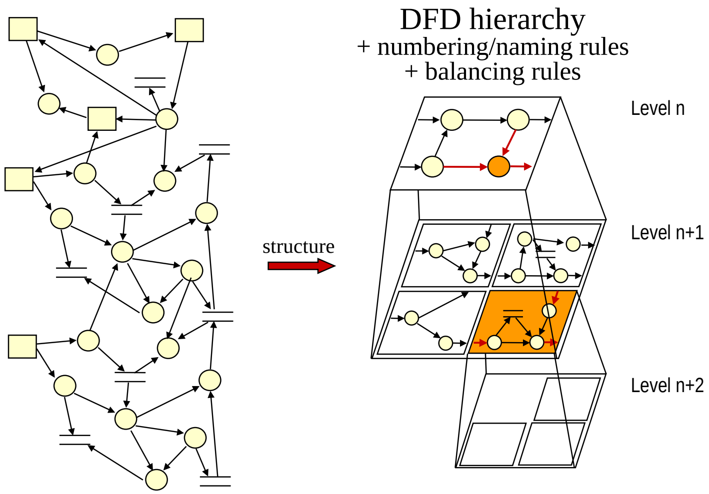


DFD Development

- Start with a *context diagram*
- Successively refine processes
- Describe all data in the data dictionary
- Describe all atomic processes by PSpecs
- Example: Order processing



DFDs--Managing Complexity



PSpecs and DD

- The format of PSpecs is not restricted
  - Free text
  - Pseudocode
- PSpecs must be defined for all atomic processes
- The format of the DD is semi-formal
- Example:

telephone number = [local extension | outside number] ← selection (or)
local extension = 2 + { number }^{3}
outside number = 0 + [local number | long distance number]
local number = prefix + access number ← composition (and)
long distance number = (1) + area code + local number ← optional
prefix = [123 | 124 | 125]
access number = { number }^{4} ← repetition
number = * any number between 0 and 9 * ← a comment

SA--Summary

- Advantages
  - Simple notation
  - Supports hierarchical decomposition
  - Easy to understand
  - Good communication medium
  - Supports consistency checks
- Disadvantages
  - Not well defined
  - No common guidelines
  - Many dialects
  - Incomplete
    - Very poor data descriptions
    - No description of control flows

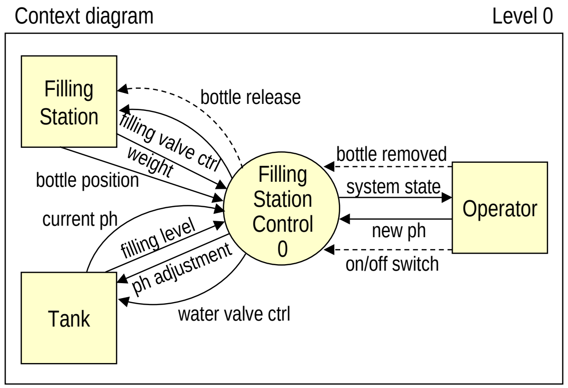
SA/RT

- Extension of SA to describe control flow
  - Activation/deactivation of processes
  - Modelling of events (signals)
  - States and state transitions
- Ward/Mellor (1985), Hatley/Pirbhai (1987)
- Additional notation (by Hatley/Pirbhai)
  - Control flow diagrams (CFDs) ← Extended DFDs
  - Process activation tables (PATs)
  - State-transition diagrams (STDs)

Idea: Each DFD contains one central control process that consumes and produces all control flows.

SA/RT--An Example

- Bottle filling station



SA/RT--Summary

- Advantages
  - Straight forward extension of SA
  - Supports hierarchical decomposition
  - Broad applicability
  - Quite well defined (STDs)
  - Tool support
- Disadvantages
  - Very poor data descriptions

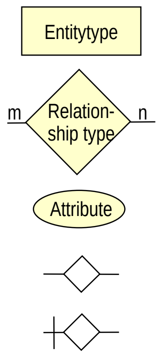
➨ Found its way to OO approaches

Data Modelling

- The entity-relationship (ER) model was developed by Chen (late 70s) to support data(base) modeling
- Focuses only on the static structure of data
- Notation
  - Entity-relationship diagrams (ERDs)
  - Attribute dictionary

ERD Notation

- According to Chen + common extensions



Set of real or abstract things about which data is stored

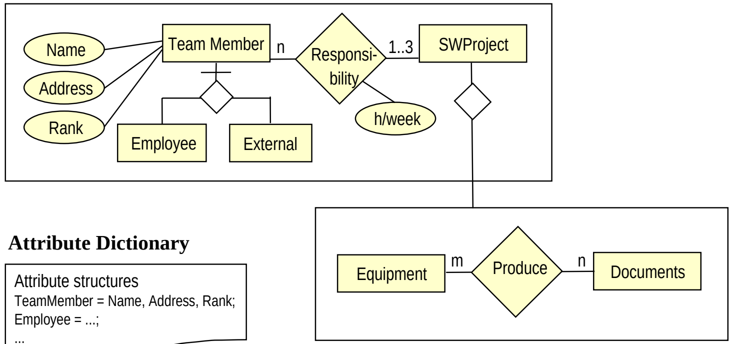
Set relations between entities with cardinalities m and n.

Information that is stored along with entities and relationships.

Composition of entities.

Classification between entity- and relationship types.

ERD--An Example



Attribute Dictionary

Attribute structures
TeamMember = Name, Address, Rank;
Employee = ...;
...

Attribute types
Name = STRING;
Address = STRING;
...

ERM--Summary

- Advantages
  - Simple notation
  - Supports hierarchical and structural decomposition
  - Easy to understand
  - Good communication medium
  - Well understood
  - Widely used
  - Good tool support
- ➨ Well-suited for DB design
- Disadvantages
  - No behaviour descriptions
  - No control descriptions
- ➨ Almost useless for non-DB applications

- ➨ Extensions of ERM lead to OO approaches

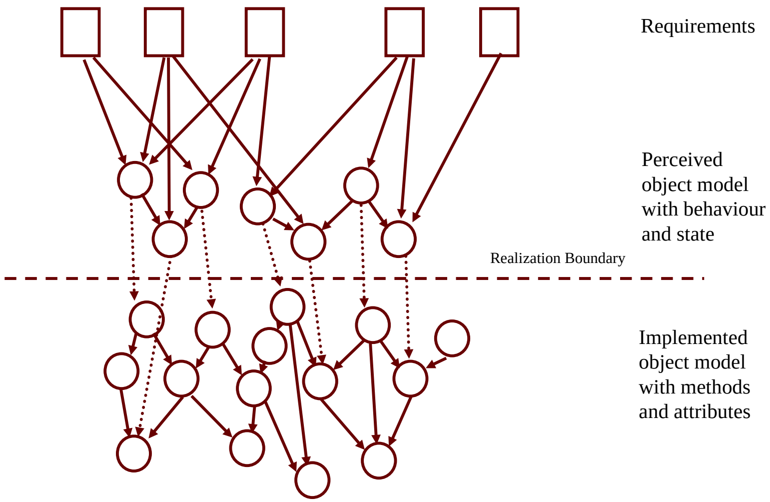
3 Important Paradigms

- Functional Orientation
- Object Orientation
- Use Case driven Object Orientation

Object Orientation

- Focus on *metaphore*: What *is* the system?
- System is structured as metaphoric "intelligent" *objects*.
- Function is provided by interacting objects.
- No separation of state and ability. Objects represent both.

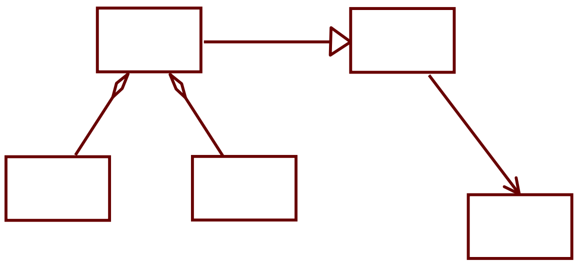
Classic Object Orientation



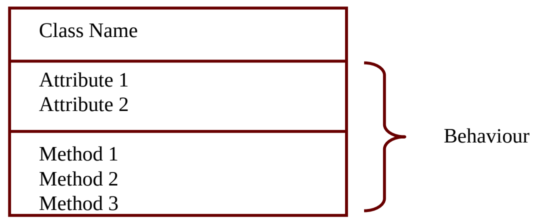
Classic OO - Simulation of reality

- Domain objects.
- Role play.
- Encapsulation, strong focus on black/white box.
- Bottom-up.
- Flat hierarchy, metaphores are not arbitrarily composable or decomposable.

Class Diagrams (UML Notation)



UML Class



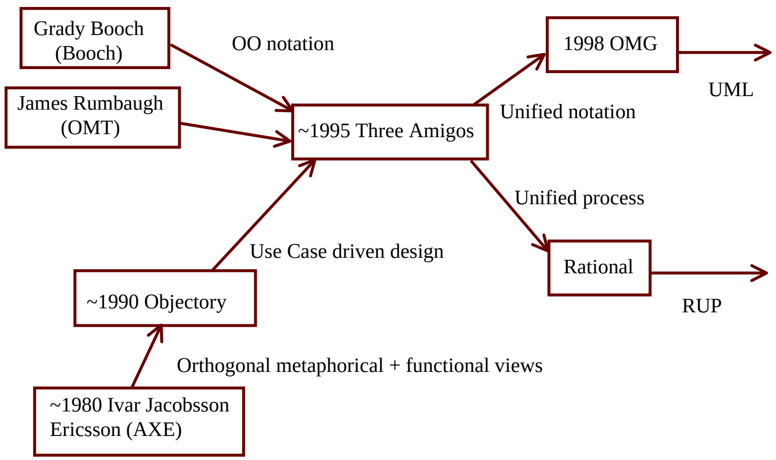
Classic OO Development algorithm

- Define requirements
- Find appropriate objects
- Map requirements onto objects
- Define methods and attributes
- Define system internal abilities
- Implement objects

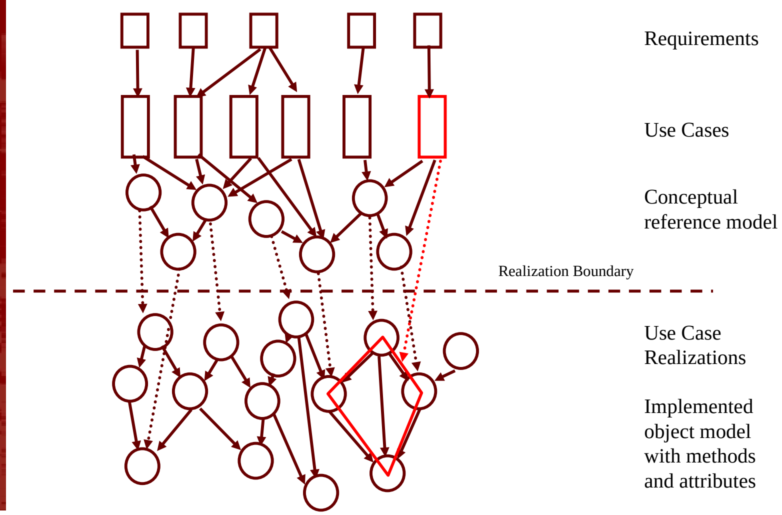
Use Case Driven Object Orientation

- Focus on both function and metaphore.
- Functionality and structure represented in orthogonal views.

History of Use Case driven design



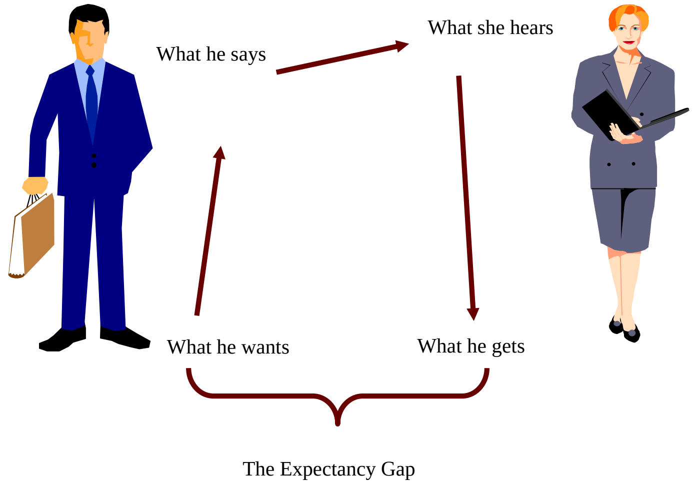
Use Case driven Object Orientation



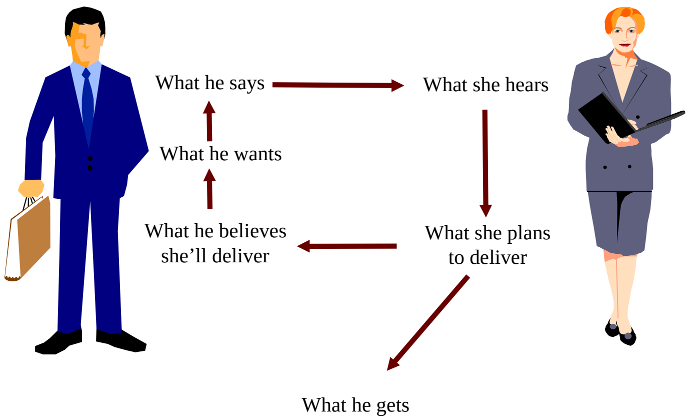
UC Driven OO Development Algorithm

- Capture requirements
- Define System Boundary in terms of Use Cases
- Define Conceptual Model
- Design Use Case Realizations
- Define Object Model
- Define internal abilities
- Implement objects

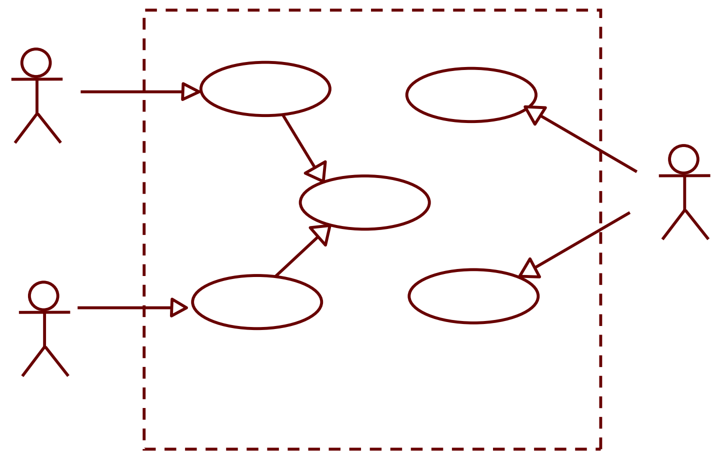
Requirement Capture



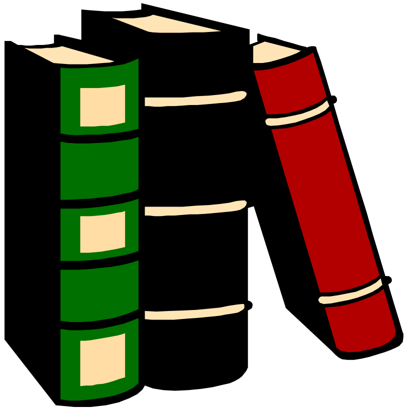
Requirement Validation



System Definition

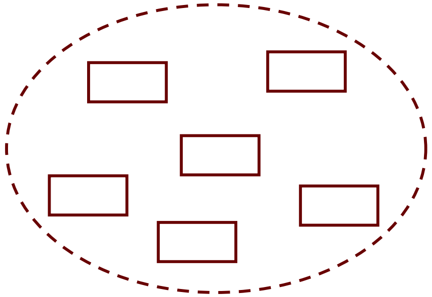


Conceptual Model



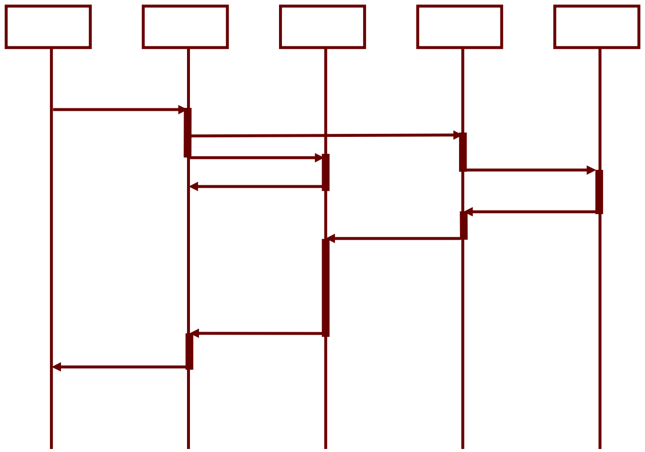
An unambiguous definition of the terminology used in the use case descriptions.

Collaborations

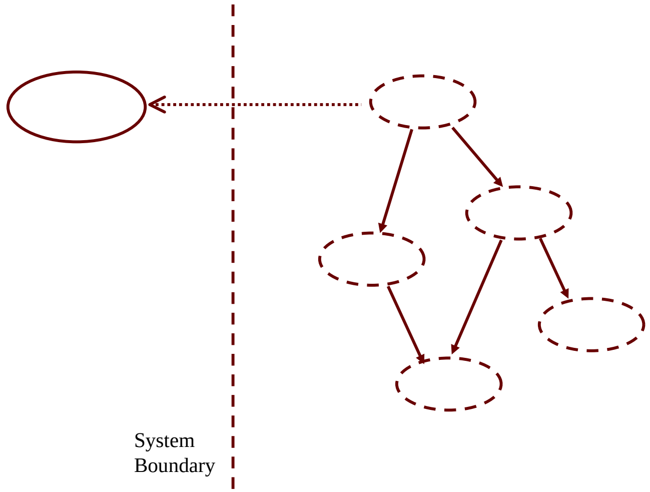


A task that is fulfilled by
a group of interacting instances

Scenarios



Functional Decomposition of Use Cases



Technical Basis for Planning: The Topology of System Abilities