

*L3 Mention Informatique
Parcours Informatique et MIAGE*

Génie Logiciel Avancé - Advanced Software Engineering

From Analysis to Design

Burkhardt Wolff

(burkhardt.wolff@universite-paris-saclay.fr)

<https://usr.lmf.cnrs.fr/~wolff>

Plan of the Chapter

- ❑ Introduction: The Role of Design
- ❑ Objectives of the Design Phase
 - capturing non-functional requirements
 - refining functional aspects
 - linking decisions, tracing requirements
- ❑ Techniques

The Role of the Design Phase

- ❑ Transition from an analysis model to a collection of more detailed, more executable, more explicit models
- ❑ Shift of Focus
 - Analysis: Understanding the Requirements Documents (Cahier de Charge)
 - Design: Understanding the Implementation and the specific constraints resulting from technology choices (programming language, frameworks, libraries, protocols, ...)
- ❑ Producing more refined UML models and documentation

The Objectives of Design (1)

- Taking « non-functional » requirements into account :
 - legal constraints, technical norms
 - security
 - performance
 - robustness
 - synchronization
- 👉 Adding technical classes and methods
- Instantiating architectural schemata
(design patterns, N-tier architectures)
- Reuse of «Components Off The Shelf » (COTS)
- for classes and packages
 - 👉 interface code might be necessary
 - 👉 component tests to provide !

The Objectives of Design (2)

- ❑ Implementing Class/Use-Case/Sequence/State-Chart/Architecture Diagrams
 - Introducing algorithmic aspects
 - Refining/detailing component interactions (interfaces)
 - Choice classes and methods implementing interactions
 - Choice of implementation language/technology
 - Coping with limitations:
- ☞ Inheritance ? Simple or multiple ?
- ☞ Visibility rules ?
- ☞ Exceptions
- ☞ Libraries ? Number Representations
(integer? longint? multi-precision?)

Refining Class Diagrams

- Adding technical classes and methods
 - arithmetic operations (int, longint, multi-precision ints ?)
 - date representations
 - classes for protocols (streams ? sockets ? VPN ? web-protocols ?)
 - classes for standard solutions
(package for credit-card payment, ...)
 - synchronization protocols for data
in distributed systems

Refining Class Diagrams

- Adding technical classes and methods
 - Reuse of «Components Off The Shelf » (COTS)
 - additional classes and operations for interface code (example: "communication layer" abstracting "POSIX", ... "data-base layer" abstracting "mySql", ...)
 - Provide tests for interfaces of COTS components to understand their behaviour in corner cases

The Objectives of Design (3)

❑ Systematics:

- Documenting the design choices
- Tracing choices wrt. requirements / cahier de charge (doors)
- Checking the coherence of choices, trying to keep the design simple
- Writing design document, linking to analyse documents

Classes of Analysis -> Design Classes

Associations of Analysis -> Attributes, methods, tables ?

Operations of Analysis -> Methods in design classes

Domain Specific Safety Standards

- ❑ Hard «digital» requirements arise:

The international standard on functional safety for software development of road vehicles ISO26262-6 requires the

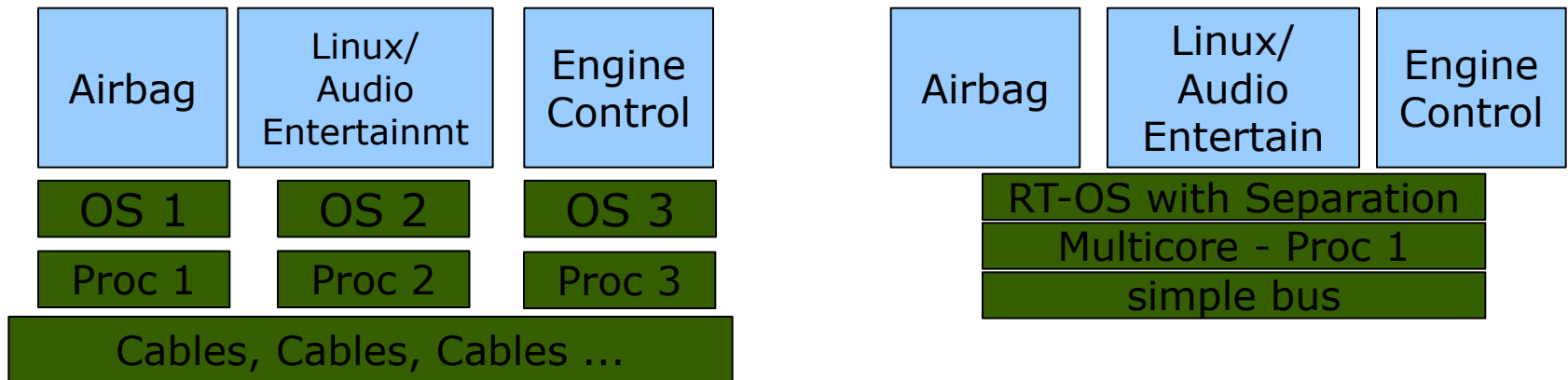
freedom from interference by software partitioning

- ❑ Thus it is aimed at providing a trusted embedded real-time operating system, which is oriented to ECUs (Electronic Control Units) in automotive industry. (avionics similarly)

Security vs. Architecture : Consequences

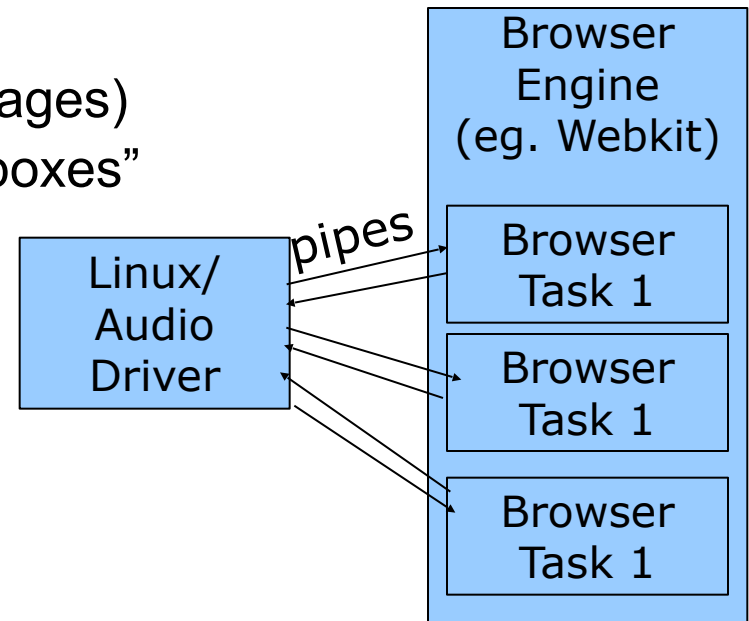
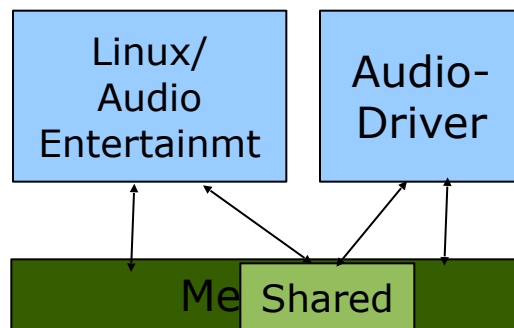
- ❑ A current industrial challenge resulting from the requirement «Freedom of interference»
 - Real-time Operating System Kernels assuring not only memory protection, but « Non-interference »

(PikeOS, Sel4, INTEGRITY-178B, RTOS Wind River Systems...)



Robustness vs. Efficiency : Consequences

- ❑ Communication between components
 - Pipe-Communication
(flexible, compatible with dynamic process creation)
 - Shared-Memory Communication
(fast, but rigid wrt. component-architecture)
 - message-passing
(very fast, but only for small messages)
 - synchronous/asynchronous “mailboxes”



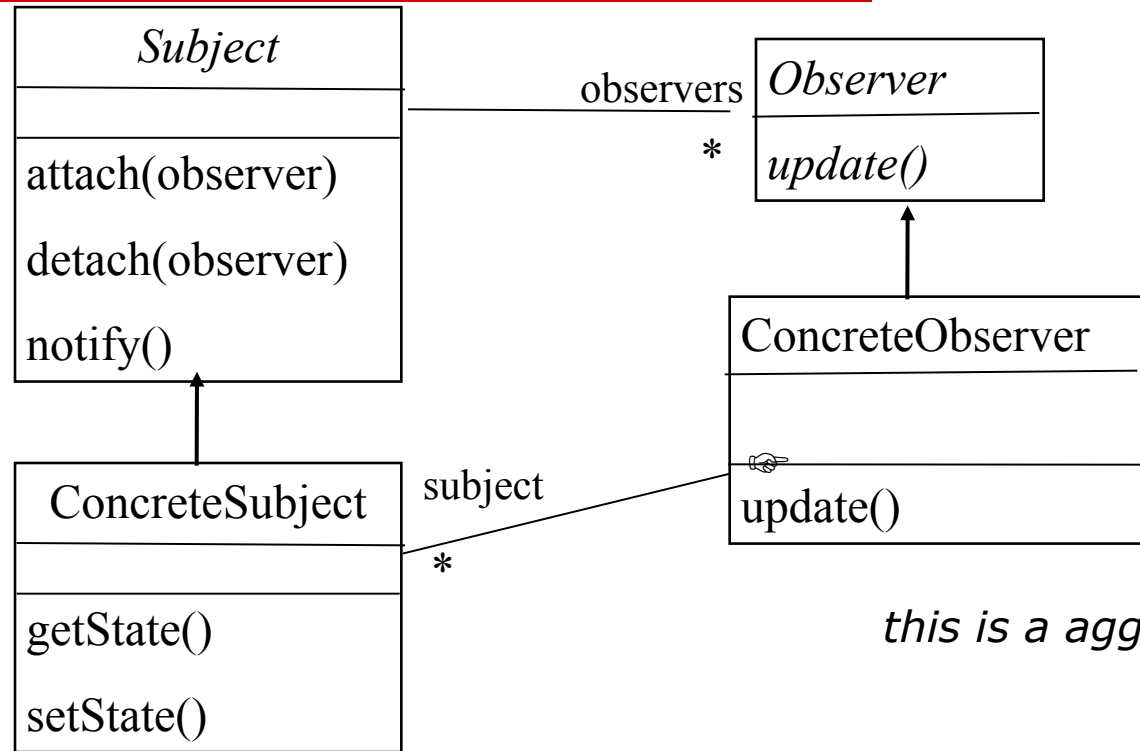
Example Design Patterns : « Observer »

- ❑ **Objective:** Maintain coherence of different « views » of a piece of data;
- ❑ **Motivation:** decoupling management of an objet and its use in different components
 - an observer can observe several objects ;
this list can dynamically change
 - an observed object can be target of several observers;
this list can dynamically change
- ❑ **Collaborations:**
 - an observer registers for the observed object
 - the observed object notifies his registerd observers
 - the observer can store specificinformation in the observed object

Example Design Patterns : « Observer »

- ❑ Directly implemented in Java :
`interface observer` where `class observable` is to derive ...
- ❑ Adding « controlers » (interactions) gives MVC.
- ❑ Classical pattern in GUI design:
some content is modified, but the gui-classes
have to update their presentation...

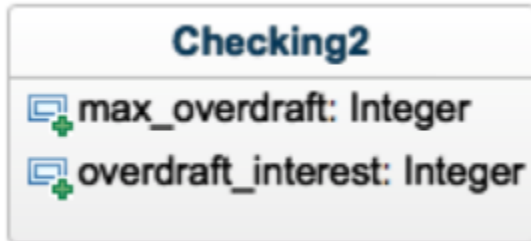
Example Design Patterns : « Observer »



this is a aggregation in practice

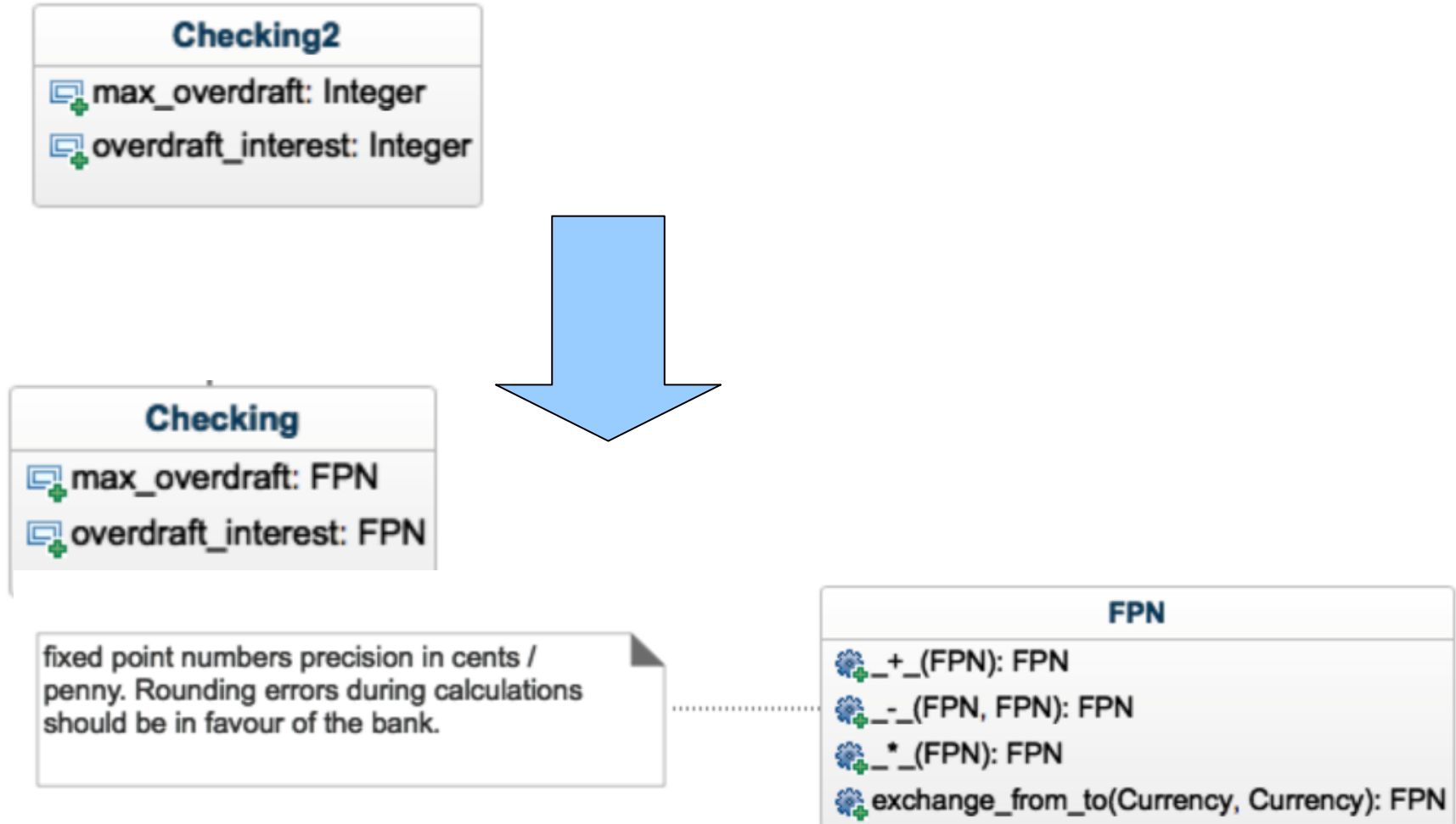
Refining Class Diagrams

- Fixing (Arithmetic) implementation types



Refining Class Diagrams

- Fixing (Arithmetic) implementation types



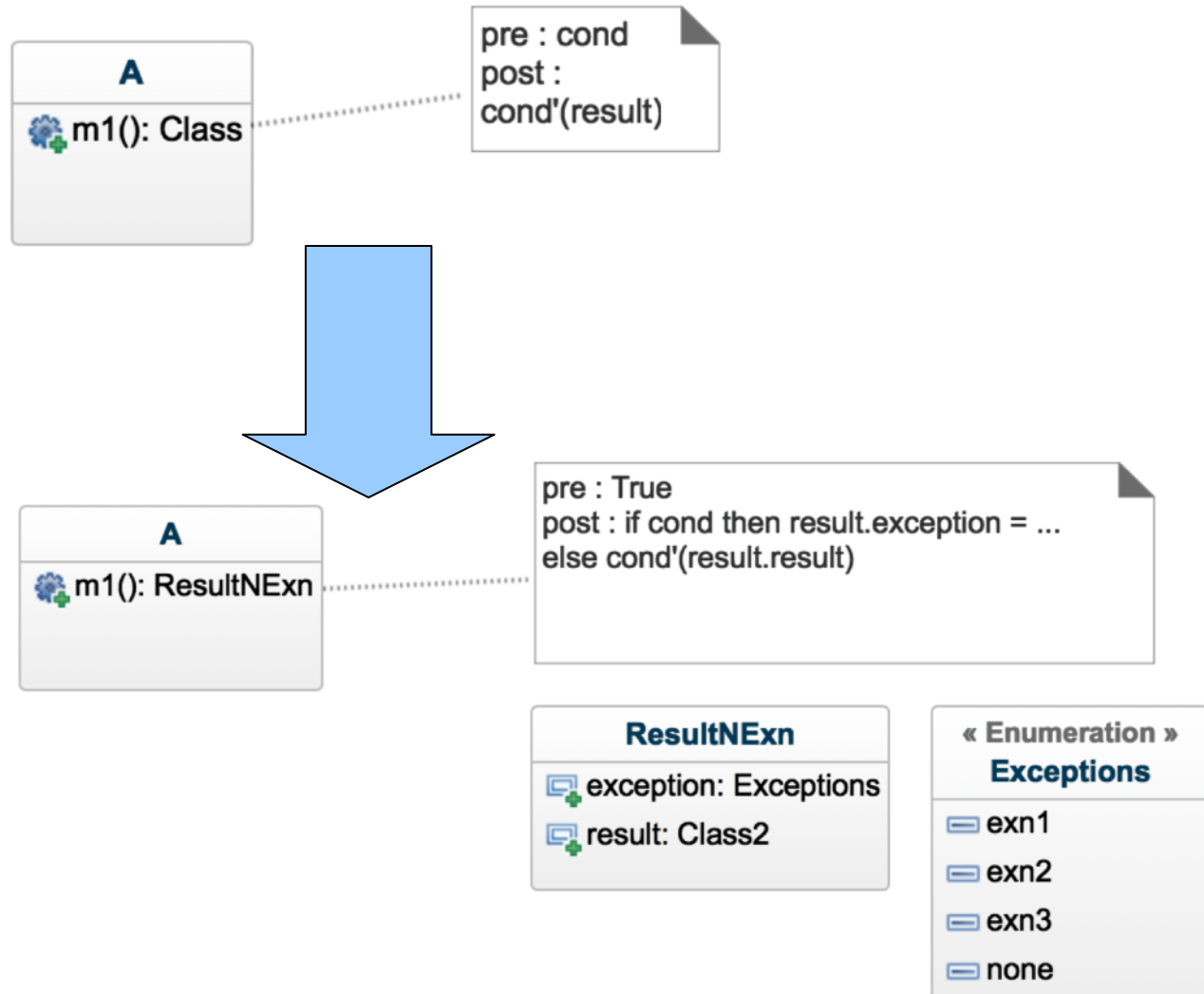
Refining Class Diagrams

- Totalizing operation contracts with exceptions



Refining Class Diagrams

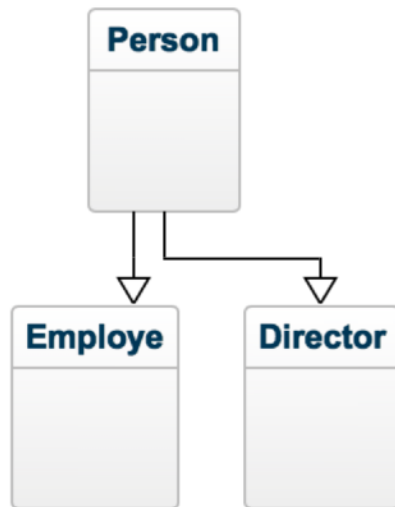
- Totalizing operation contracts with exceptions



Refining Class Diagrams

➤ Expressing Inheritance

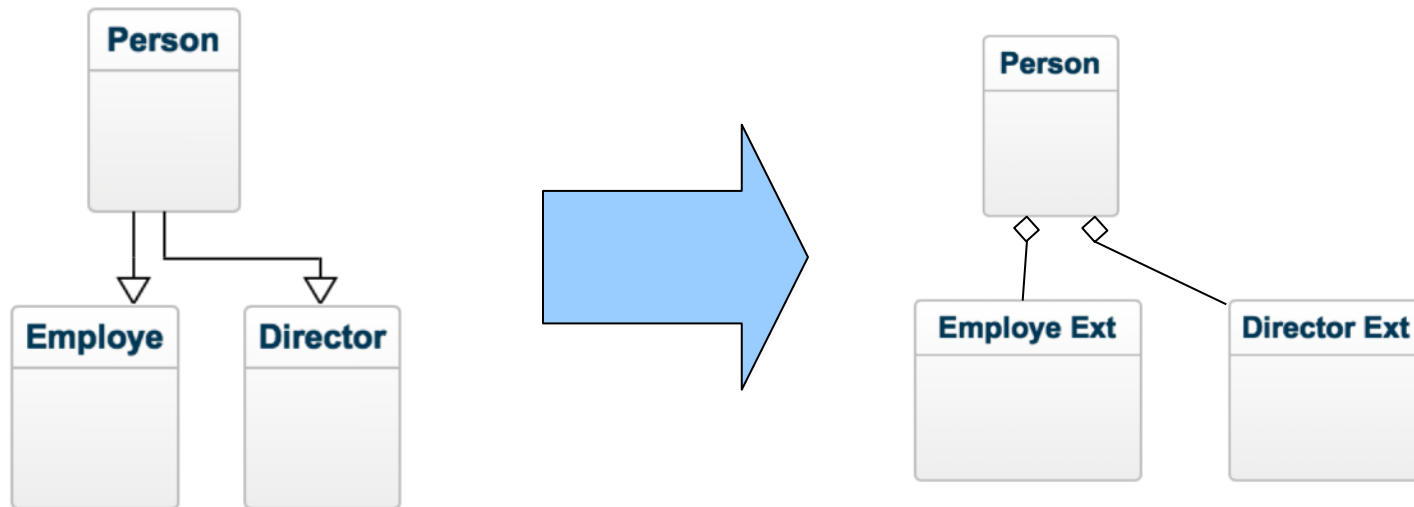
- ... because the target language doesn't support it
- ... because the instance shouldn't lose its identity when changes



Refining Class Diagrams

➤ Expressing Inheritance

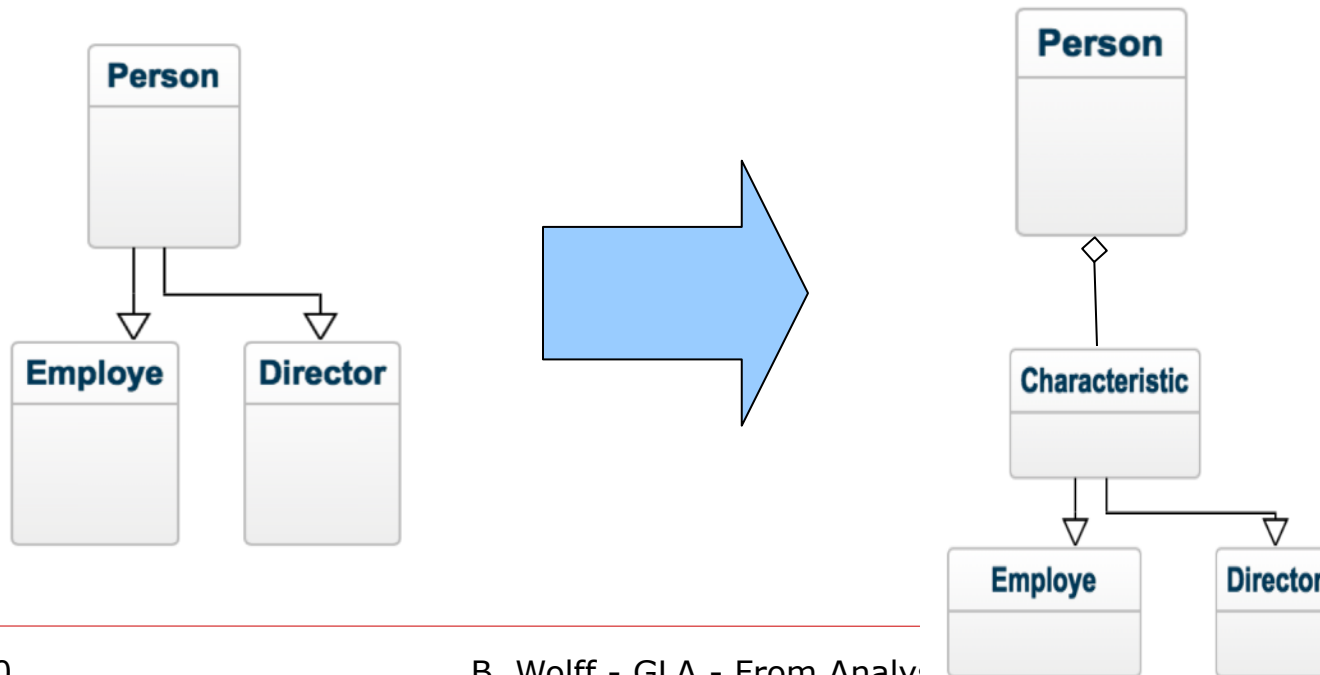
- ... because the target language doesn't support it
- ... because the instance shouldn't lose its identity when changes



Refining Class Diagrams

➤ Expressing Inheritance

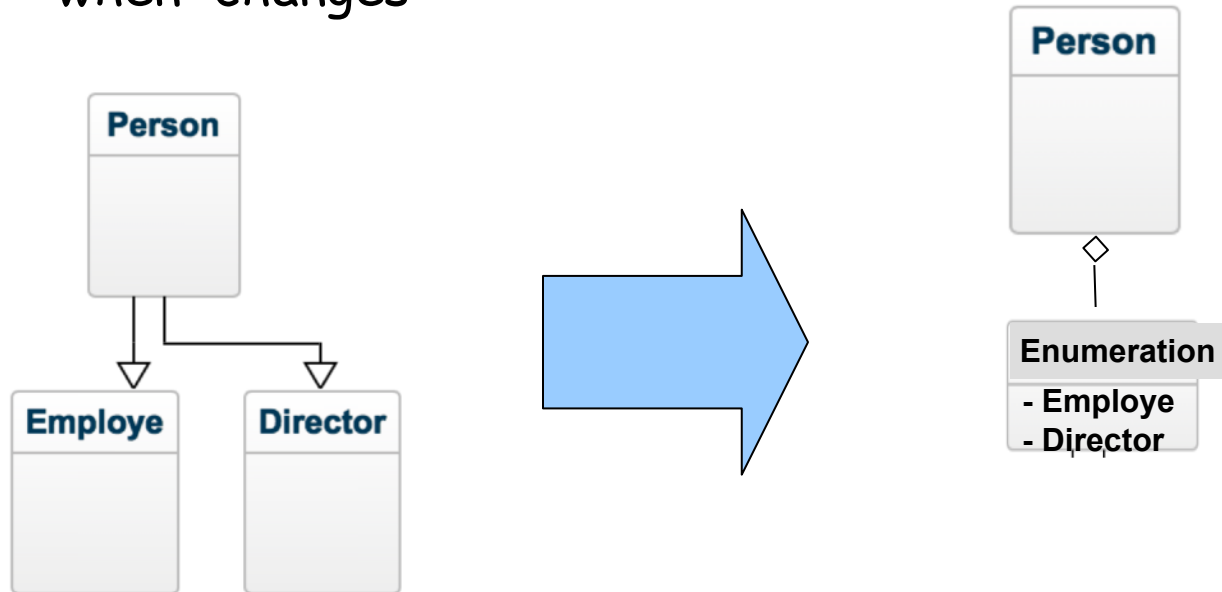
- ... because the target language doesn't support it
- ... because the instance shouldn't lose its identity when changes



Refining Class Diagrams

➤ Expressing Inheritance

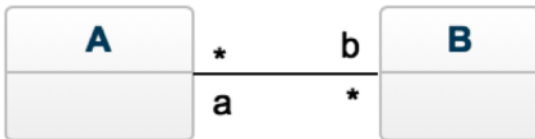
- ... because the target language doesn't support it
- ... because the instance shouldn't lose its identity when changes



Refining Class Diagrams

➤ Implementing Associations

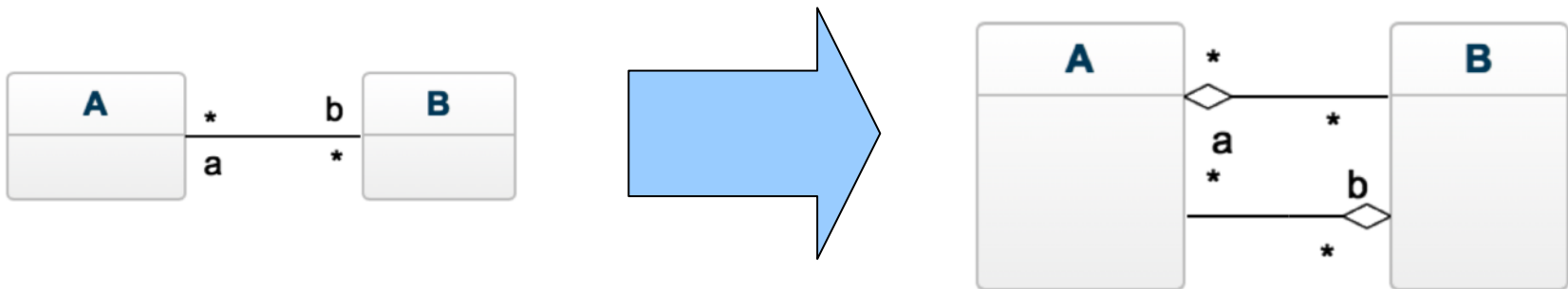
- ... depends on cardinality (1 ? * ? 1..5 ?)
- ... depends on type (set ? multiset ? list ?)



Refining Class Diagrams

➤ Implementing Associations

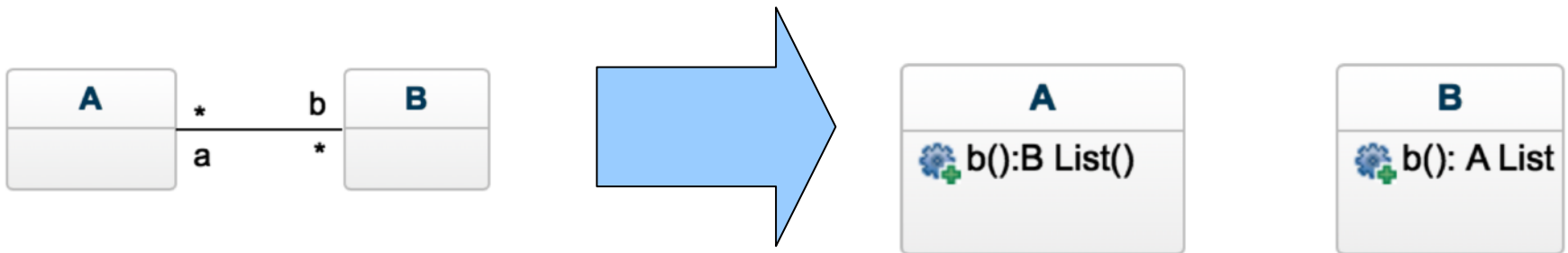
- ... depends on cardinality (1 ? * ? 1..5 ?)
- ... depends on type (set ? multiset ? list ?)
- ... as mutually linked lists (or arrays) of references



Refining Class Diagrams

➤ Implementing Associations

- ... depends on cardinality (1 ? * ? 1..5 ?)
- ... depends on type (set ? multiset ? list ?)
- ... as recomputing methods ...



Refining Class Diagrams

➤ Implementing Associations

- ... depends on cardinality (1 ? * ? 1..5 ?)
- ... depends on type (set ? multiset ? list ?)
- ... as recomputing methods using an index table

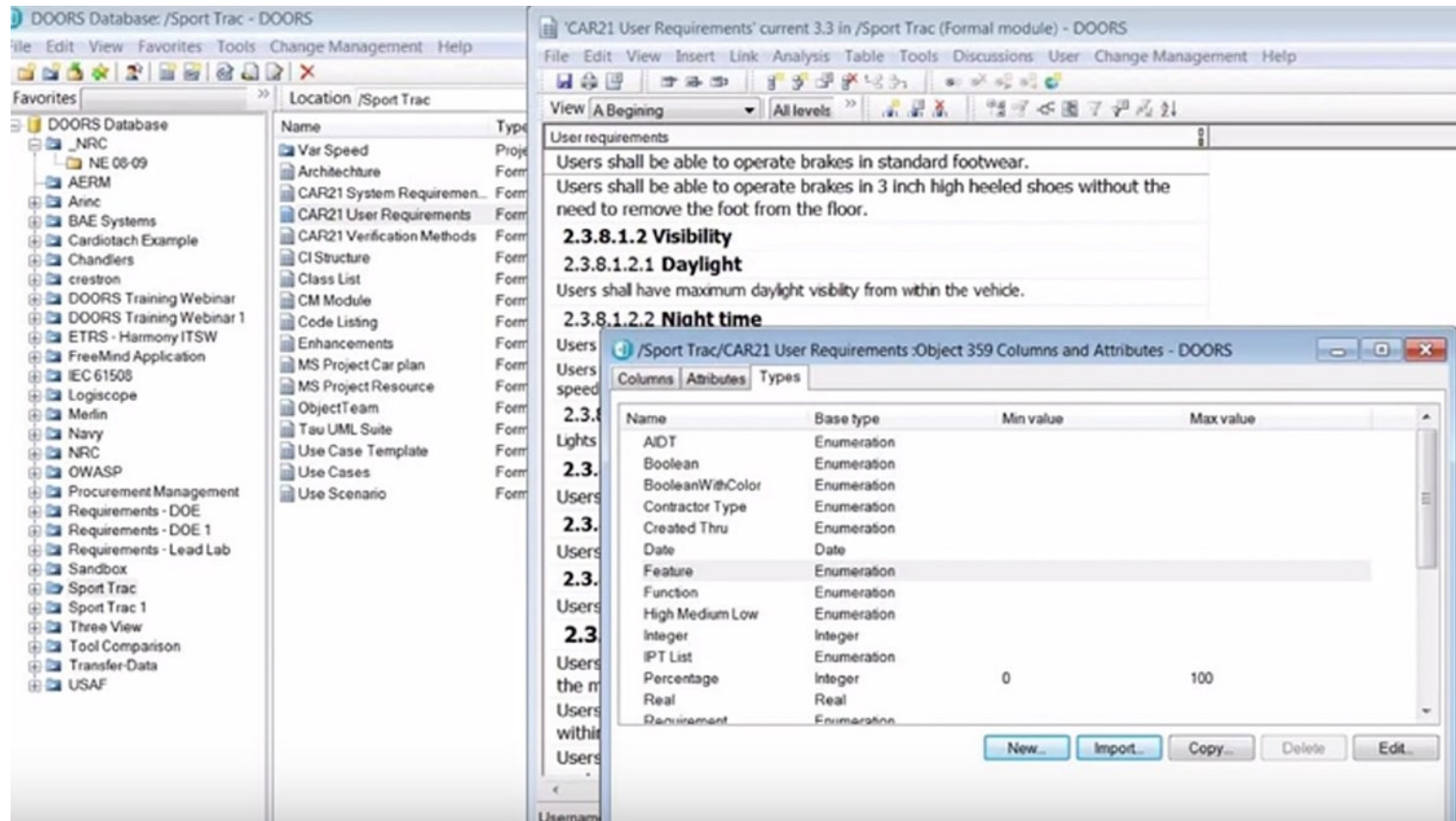


Tracing Requirements

- ❑ Tracing requirements from CDC over Analysis and Design Milestones is mandatory in many certification processes
- ❑ Technical Solution:
 - Rational Dynamic Object Oriented Requirements System (DOORS) client-server application, with a Windows-only client and servers for Linux, Windows, and Solaris.
 - There is also a web client, DOORS Web Access.
 - For example, it is common practice to capture verification relationships to demonstrate that a requirement is verified by a certain test artefact.
 - DOORS comes with an own modeling language allowing to generate UML diagrams
 - <https://www.ibm.com/de-de/marketplace/requirements-management/details>

Tracing Requirements

❑ DOORS screenshot



Conclusion

- ❑ Refinement of the Analysis docs
- ❑ Objectives of the Design Phase
 - capturing non-functional requirements
 - refining functional aspects
 - linking decisions, tracing requirements
- ❑ Techniques numerous, and depend on chosen target languages / technologies