L3 Mention Informatique
Parcours Informatique et MIAGE

Génie Logiciel Avancé - Advanced Software Engineering

From Analysis to Design

Burkhart Wolff
wolff@lri.fr

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 dor 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
  - 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

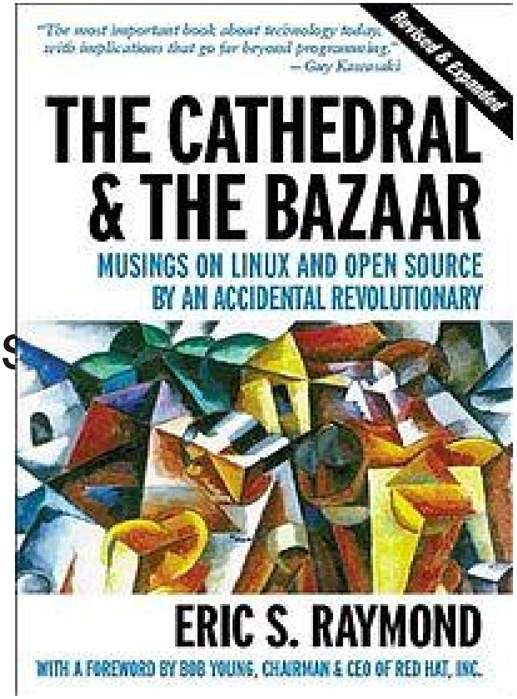
Classes of Analysis -> Design Classes
Associations of Analysis -> Attributes, methods, tables ?
Operations of Analysis -> Methods in design classes

Context: Norms in Software Engineering

Amusing Book: Raymonds Cathedral-Bazaar

Metaphor for (Open-Source) Processes:

- ... The *Cathedral* model, in which source code is available with each software release, but code developed between releases is restricted to an exclusive group of software developers GNU Emacs and GCC are presented as examples.
- ... The *Bazaar* model, in which the code is developed over the Internet in view of the public. Raymond credits Linus Torvalds, leader of the Linux kernel project, as the inventor of this process.


"The most important book about technology today, with implications that go far beyond programming."
– Guy Kawasaki
Revised & Expanded
THE CATHEDRAL & THE BAZAAR
MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY
ERIC S. RAYMOND
WITH A FOREWORD BY BOB YOUNG, CHAIRMAN & CEO OF RED HAT, INC.

Norms for Cathedral Style

- Many attempts to control development processes and software products by standards (norms)
- Attempts to assure and certify software quality.
  - Most serious and relevant (in France):
  - DO 178B (Avionics)
  - ISO/IEC/IEEE 29119 (Software Test)
  - ISO/IEC/IEEE 15408 «Common Criteria» for computer security certification requiring formal models as well as proof techniques for EAL 6 and EAL 7.

Domain Specific Safety Standards

- The following standards use SIL as a measure of reliability and/or risk reduction
  - ANSI/ISA S84 (Functional safety of safety instrumented systems for the process industry sector)
  - IEC EN 61508 (Functional safety of electrical/electronic/programmable electronic safety related systems)
  - IEC 61511 (Safety instrumented systems for the process industry sector)
  - IEC 61513 (Nuclear Industry)
  - IEC 62061 (Safety of machinery)
  - EN 50128 (Railway applications - Software for railway control and protection)
  - EN 50129 (Railway applications - Safety related electronic systems for signalling
  - EN 50402 (Fixed gas detection systems)

Domain Specific Safety Standards

- Hard «digital» requirements arise:

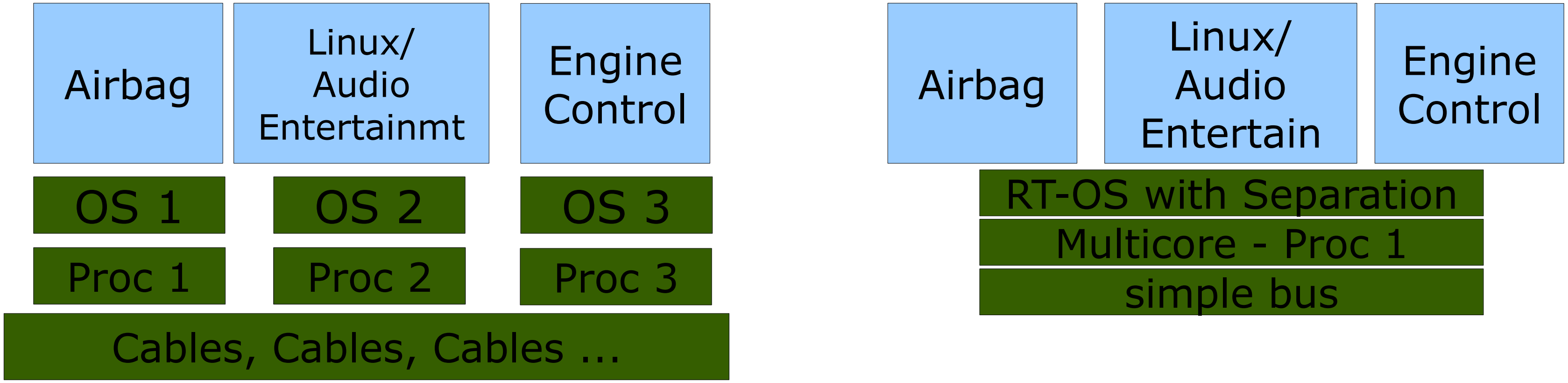
  The international standard on functional safety for software development of road vehicles ISO26262-6 requires the

  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)

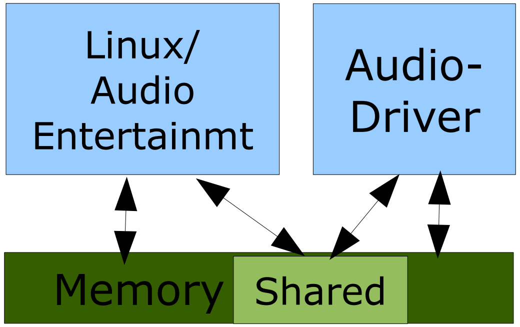
Security vs. Architecture : Consequences

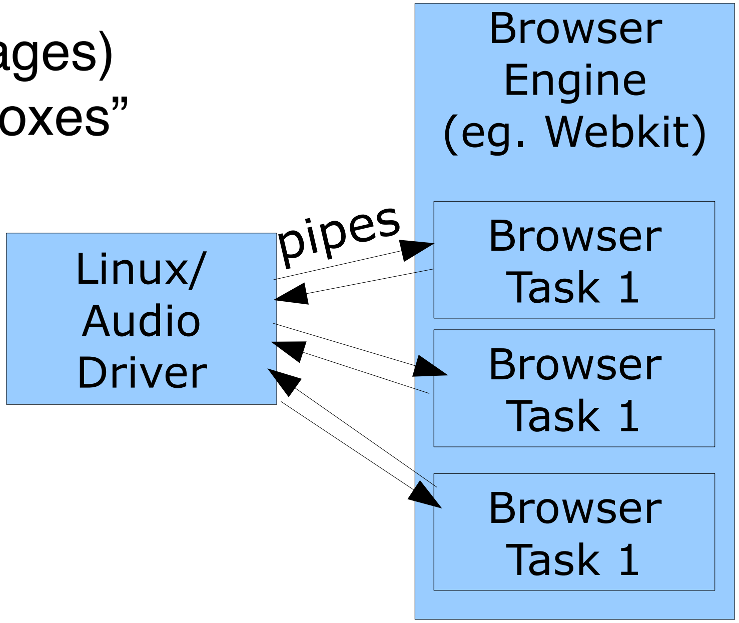
- A current industrial challenge resulting from the requirement «Freedom of interference»
  - Real-time Operating System Kernels assuring not only memory protection, but « Non-interference »
    (PikeOS, Sel4, INTEGRITY-178B, RTOS Wind River Systems…)



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"



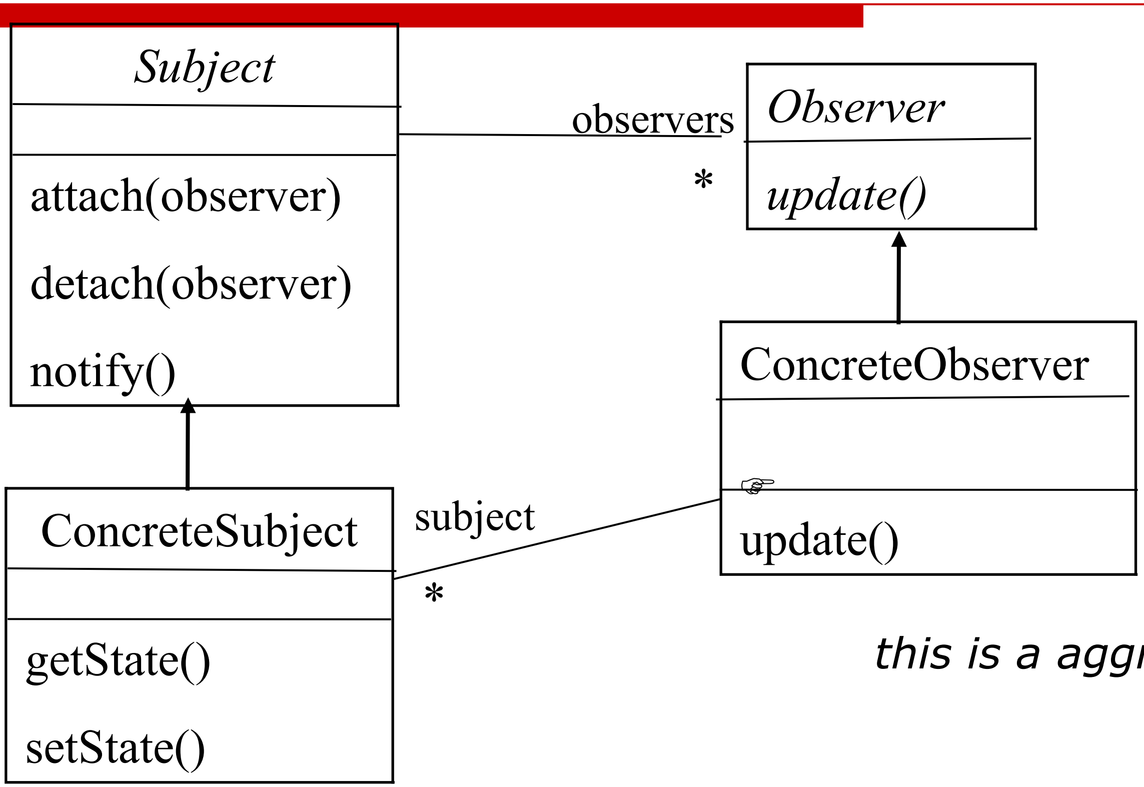


Example Design Patterns : « Observer »

- **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 »



this is a aggregation in practice

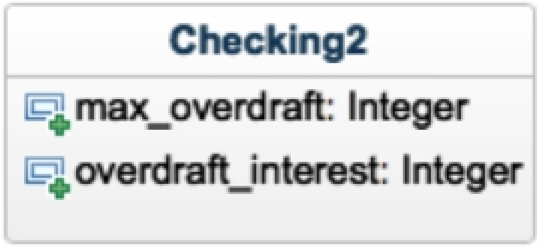
- Directly impemented in Java :
  `interface` `observer` where class `observable` is to derive …
- Adding « controlers » (interactions) gives MVC.

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
  - 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

Refining Class Diagrams

- Fixing (Arithmetic) implementation types



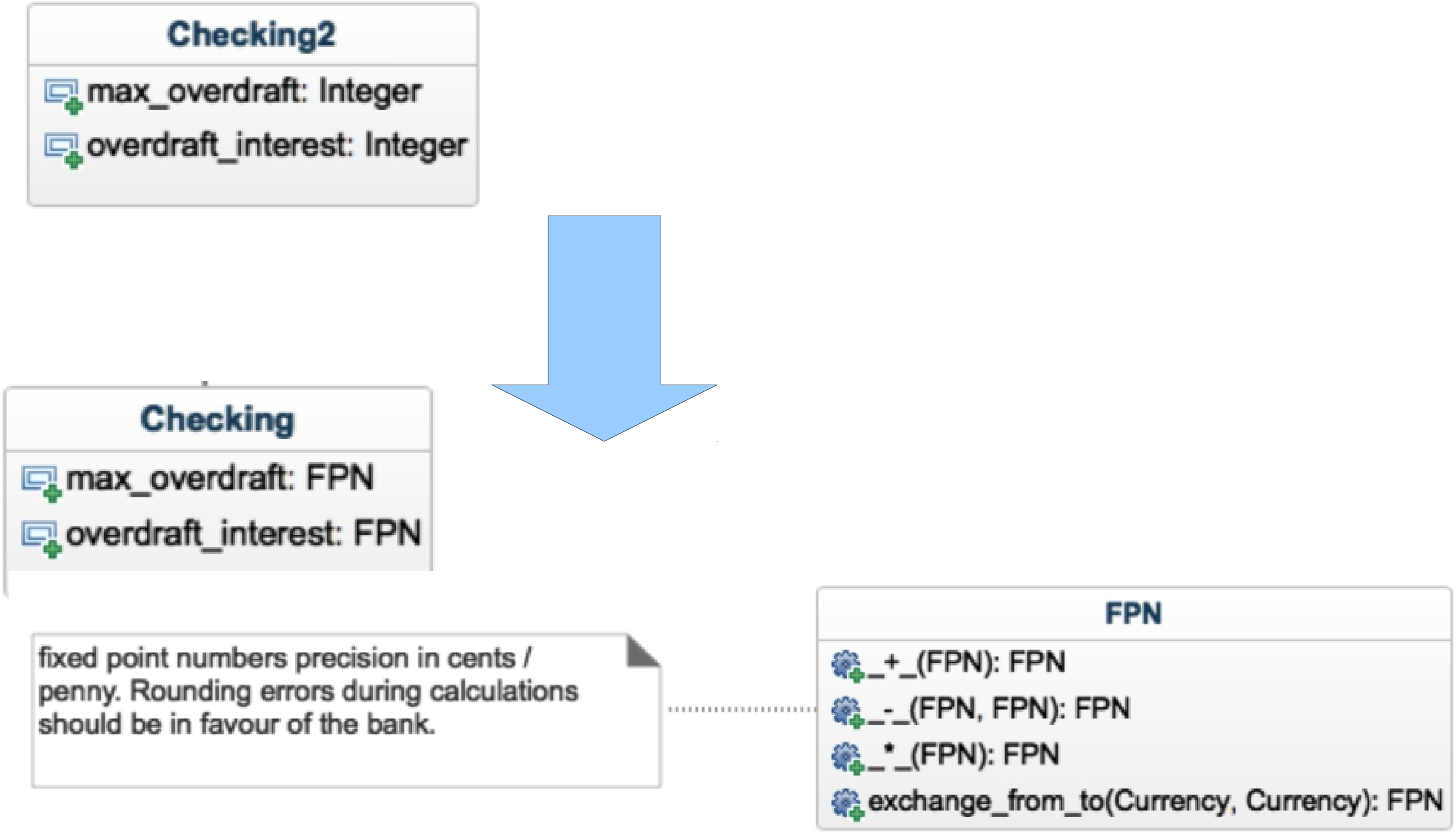
Checking2

max_overdraft: Integer

overdraft_interest: Integer

Refining Class Diagrams

- Fixing (Arithmetic) implementation types



Checking2

max_overdraft: Integer

overdraft_interest: Integer

Checking

max_overdraft: FPN

overdraft_interest: FPN

fixed point numbers precision in cents / penny. Rounding errors during calculations should be in favour of the bank.

FPN

_+_(FPN): FPN

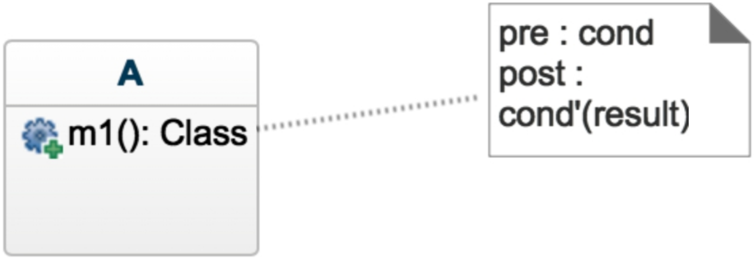
_-_(FPN, FPN): FPN

_*_(FPN): FPN

exchange_from_to(Currency, Currency): FPN

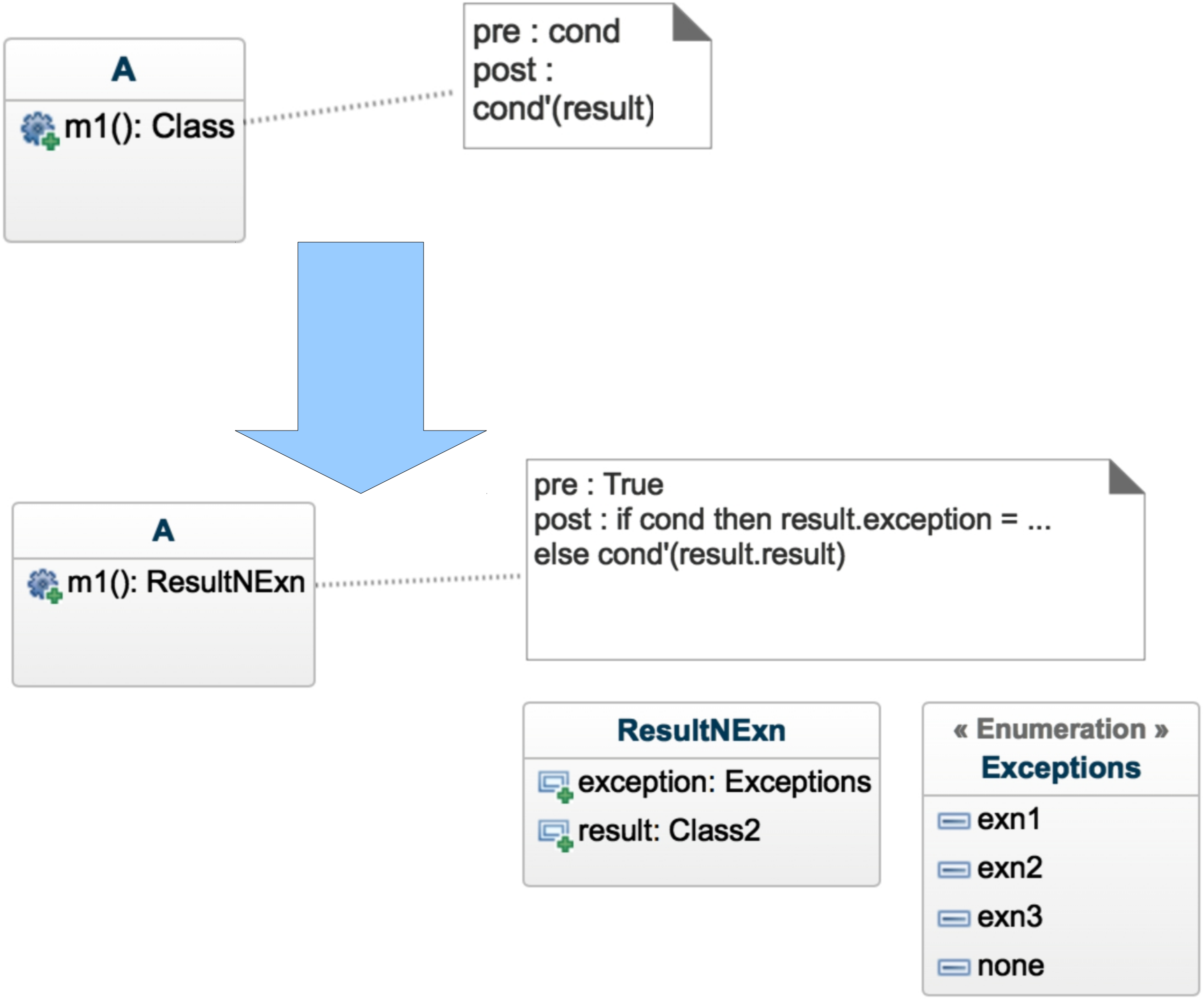
Refining Class Diagrams

- Totalizing operation contracts with exceptions



Refining Class Diagrams

- Totalizing operation contracts with exceptions



A

m1(): Class

pre : cond
post :
cond'(result)

A

m1(): ResultNExn

pre : True
post : if cond then result.exception = ...
else cond'(result.result)

ResultNExn

exception: Exceptions

result: Class2

« Enumeration »
Exceptions

exn1

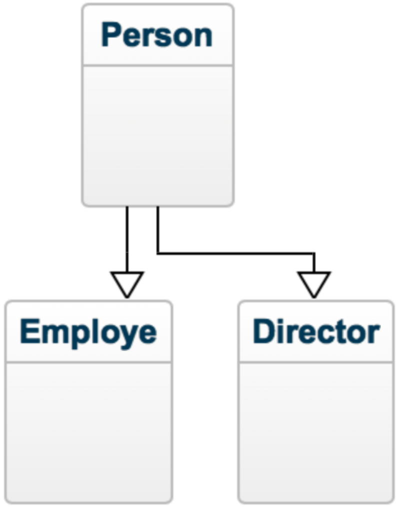
exn2

exn3

none

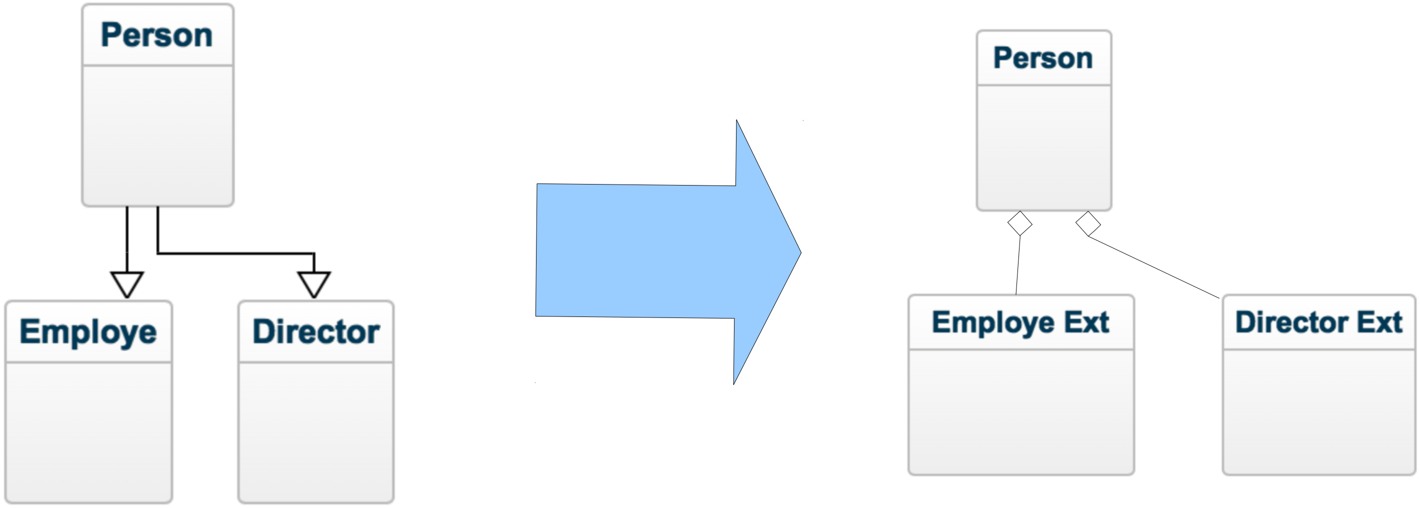
Refining Class Diagrams

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



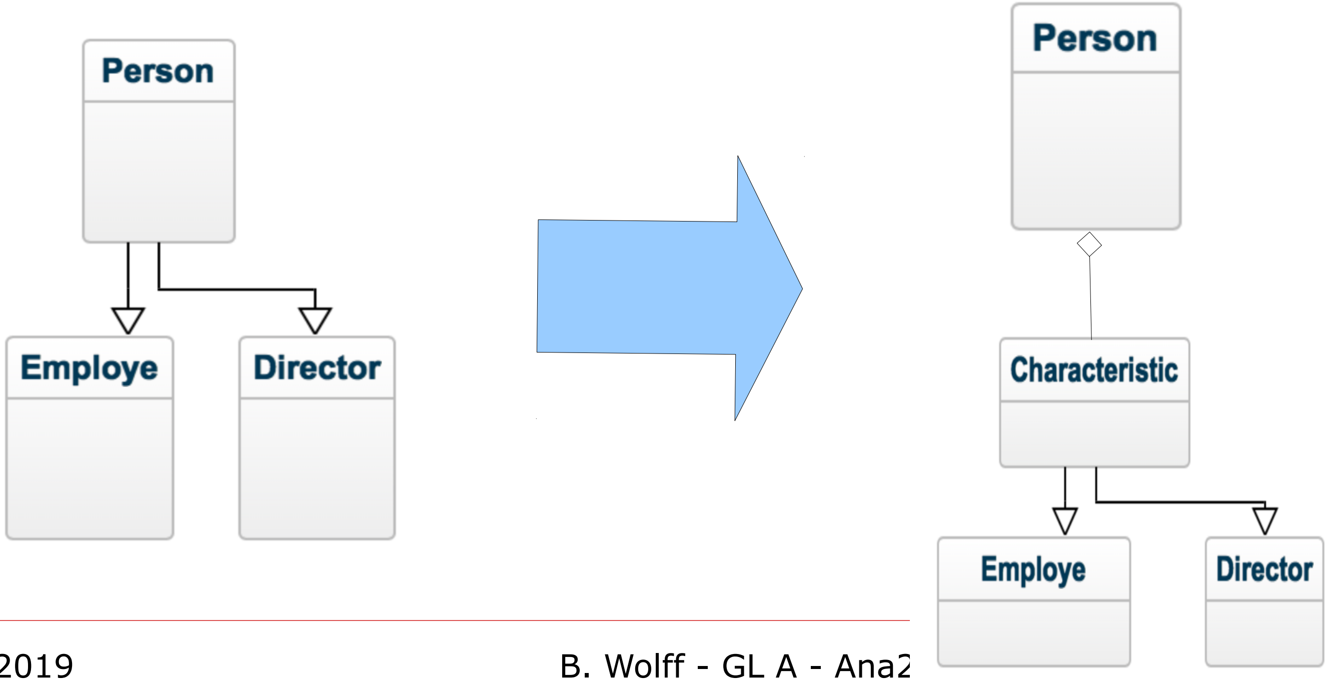
Refining Class Diagrams

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



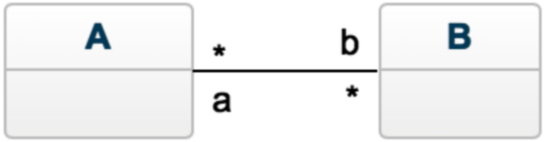
Refining Class Diagrams

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



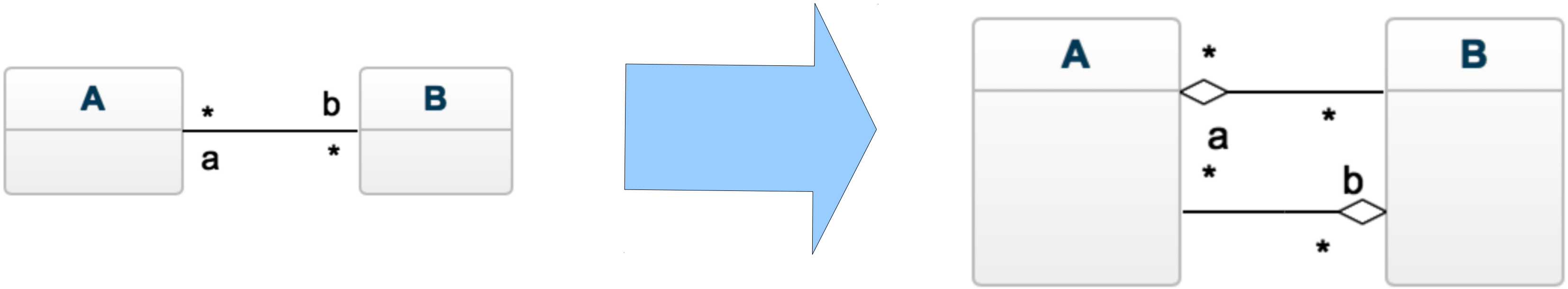
Refining Class Diagrams

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



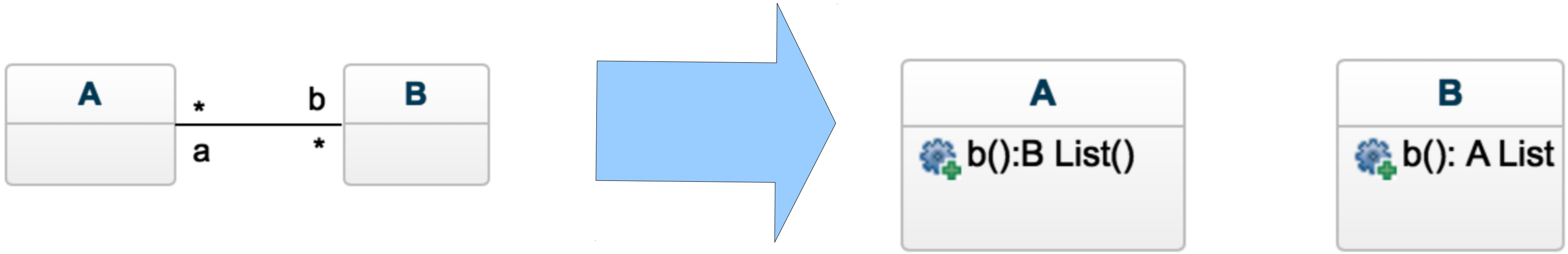
Refining Class Diagrams

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



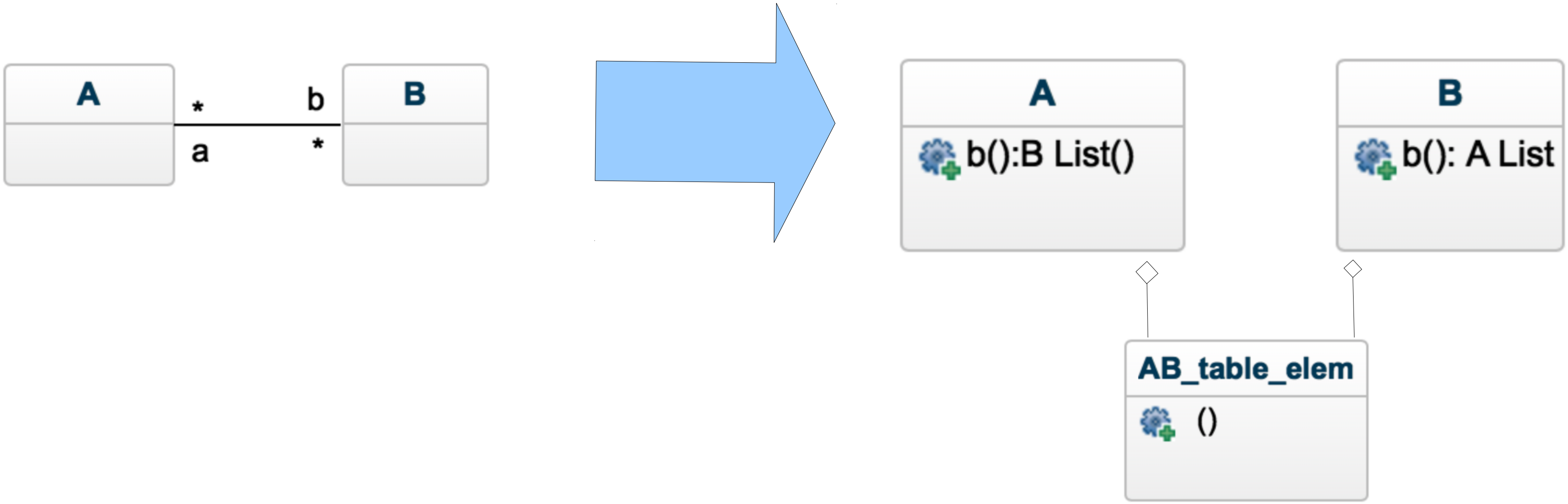
Refining Class Diagrams

- 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

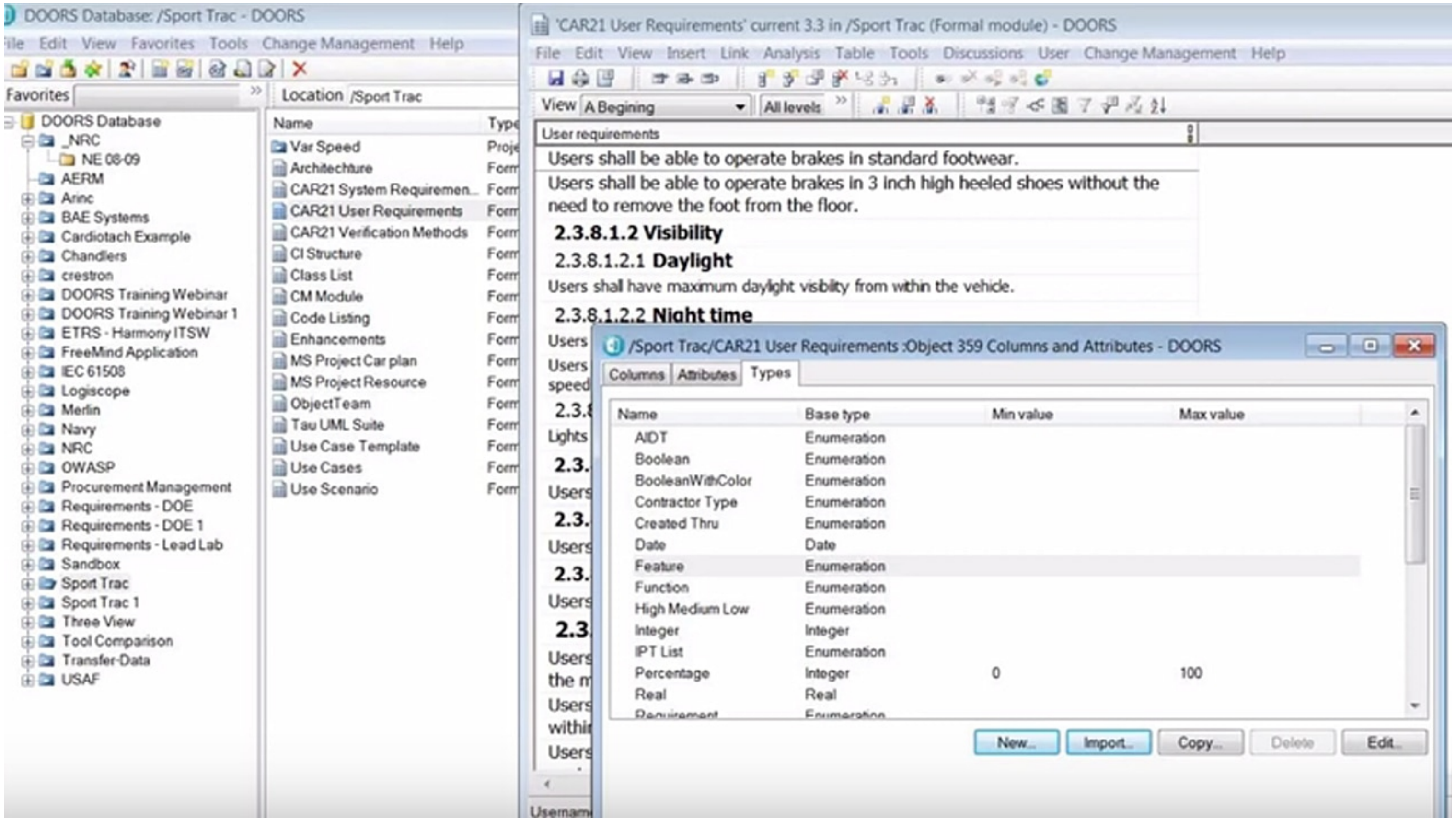


Tracing Requirements

- 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


DOORS Database: /Sport Trac - DOORS
ile Edit View Favorites Tools Change Management Help
Favorites
Location /Sport Trac
'CAR21 User Requirements' current 3.3 in /Sport Trac (Formal module) - DOORS
File Edit View Insert Link Analysis Table Tools Discussions User Change Management Help
View A Begining
All levels
/Sport Trac/CAR21 User Requirements :Object 359 Columns and Attributes - DOORS
Columns
Atributes
Types
New...
Import...
Copy...
Delete
Edit...
Usernam

- DOORS Database
  - _NRC
    - NE 08-09
  - AERM
  - Arinc
  - BAE Systems
  - Cardiotach Example
  - Chandlers
  - crestron
  - DOORS Training Webinar
  - DOORS Training Webinar 1
  - ETRS - Harmony ITSW
  - FreeMind Application
  - IEC 61508
  - Logiscope
  - Merlin
  - Navy
  - NRC
  - OWASP
  - Procurement Management
  - Requirements - DOE
  - Requirements - DOE 1
  - Requirements - Lead Lab
  - Sandbox
  - Sport Trac
  - Sport Trac 1
  - Three View
  - Tool Comparison
  - Transfer-Data
  - USAF

Name	Type
Var Speed	Proje
Architechture	Form
CAR21 System Requiremen...	Form
CAR21 User Requirements	Form
CAR21 Verification Methods	Form
CI Structure	Form
Class List	Form
CM Module	Form
Code Listing	Form
Enhancements	Form
MS Project Car plan	Form
MS Project Resource	Form
ObjectTeam	Form
Tau UML Suite	Form
Use Case Template	Form
Use Cases	Form
Use Scenario	Form

User requirements

Users shall be able to operate brakes in standard footwear.

Users shall be able to operate brakes in 3 inch high heeled shoes without the need to remove the foot from the floor.

2.3.8.1.2 Visibility

2.3.8.1.2.1 **Daylight**

Users shall have maximum daylight visiblity from within the vehicle.

2.3.8.1.2.2 **Night time**

Name	Base type	Min value	Max value
AIDT	Enumeration		
Boolean	Enumeration		
BooleanWithColor	Enumeration		
Contractor Type	Enumeration		
Created Thru	Enumeration		
Date	Date		
Feature	Enumeration		
Function	Enumeration		
High Medium Low	Enumeration		
Integer	Integer		
IPT List	Enumeration		
Percentage	Integer	0	100
Real	Real		
Requirement	Enumeration		

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