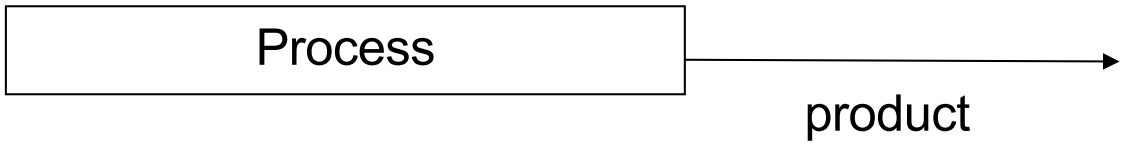
Requirement Engineering

SoftEng
http://softeng.polito.it

RE

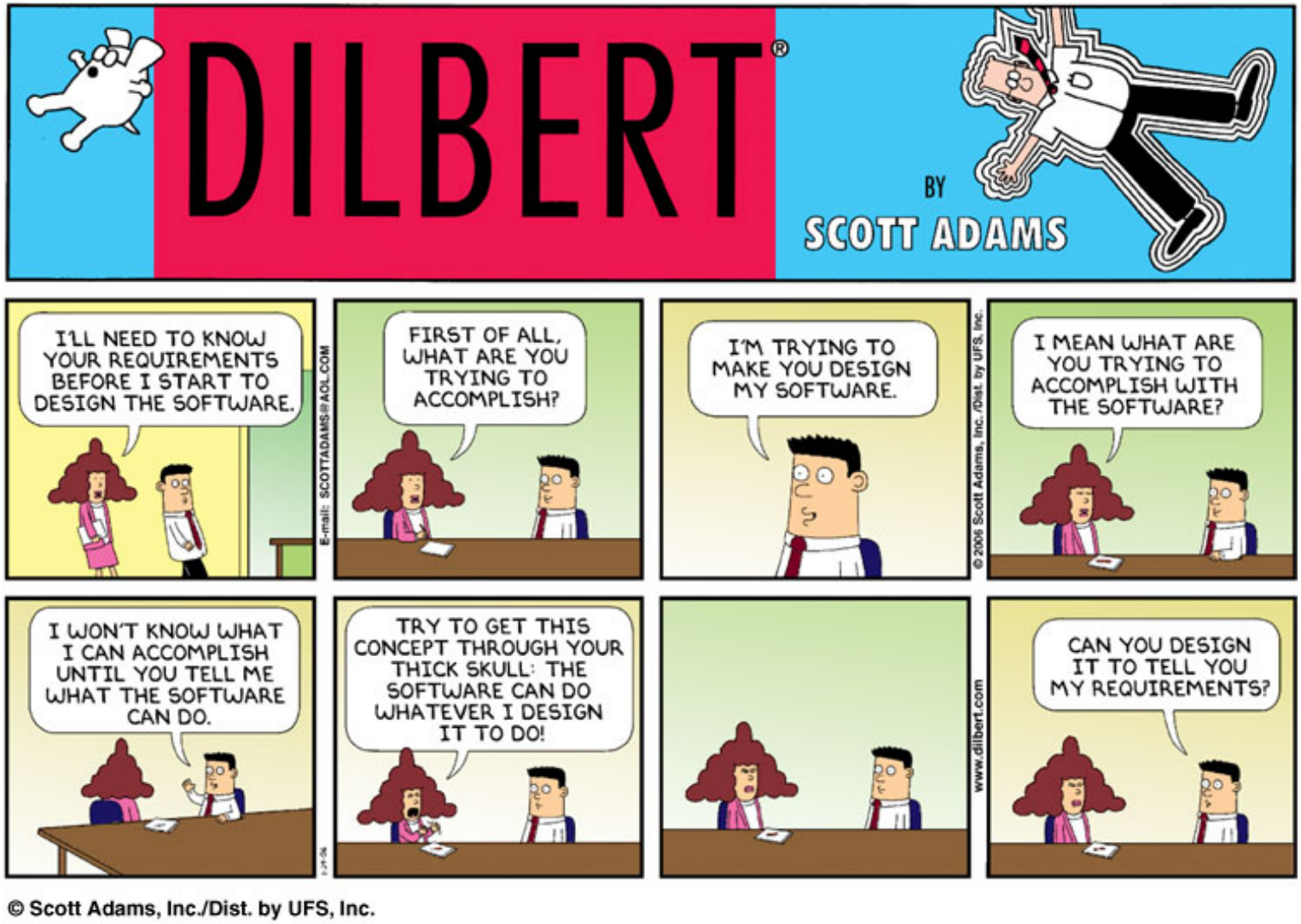
- Requirement engineering is about defining the product properties before starting development



Product properties

- Functional
  - what should the program do
- Non functional
  - Modalities applied to functions offered
    - Reliability
    - Usability
    - Performance
    - Maintainability
    - Security
    - ...

RE



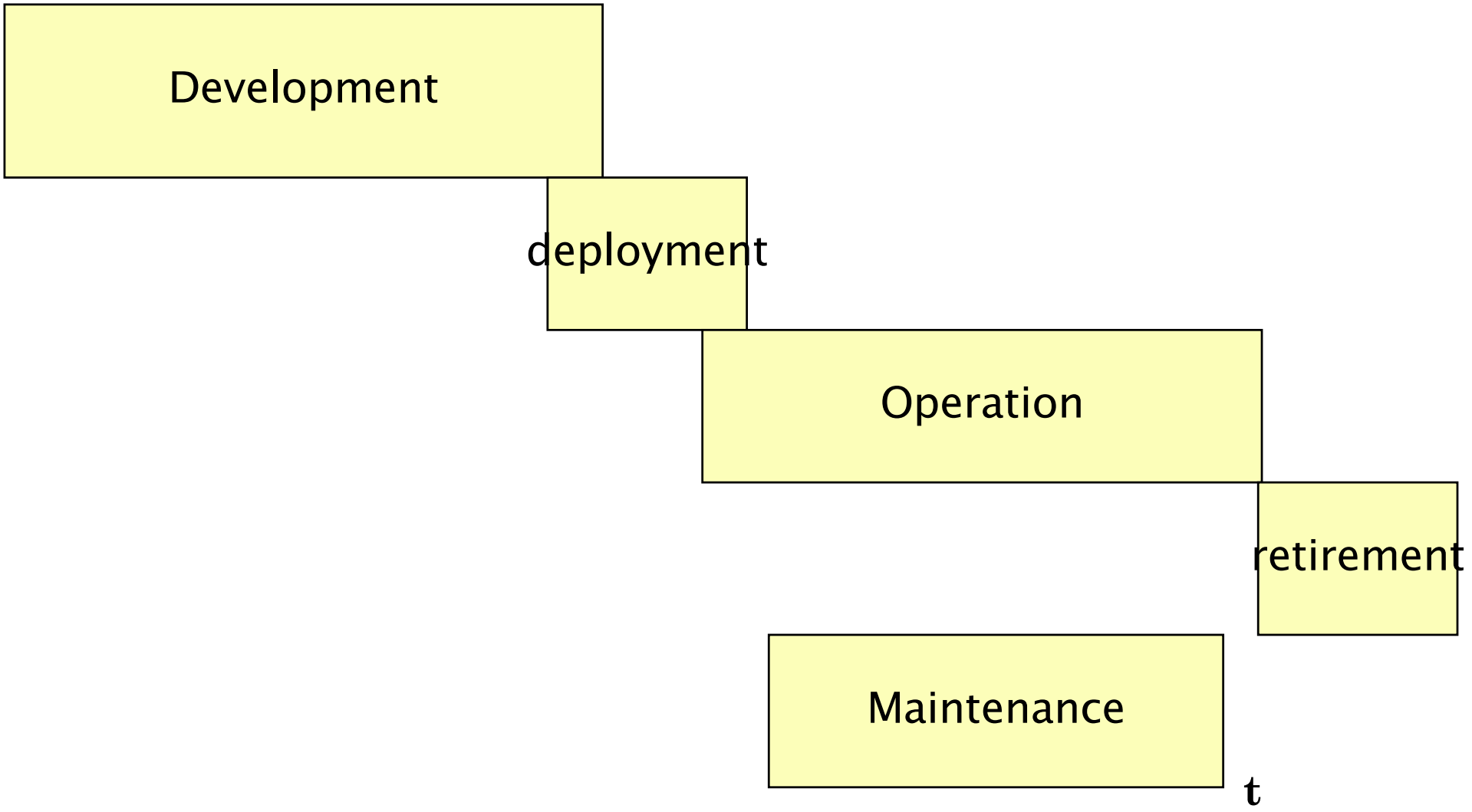
Without RE

- Product properties are unclear
- Testing is not feasible

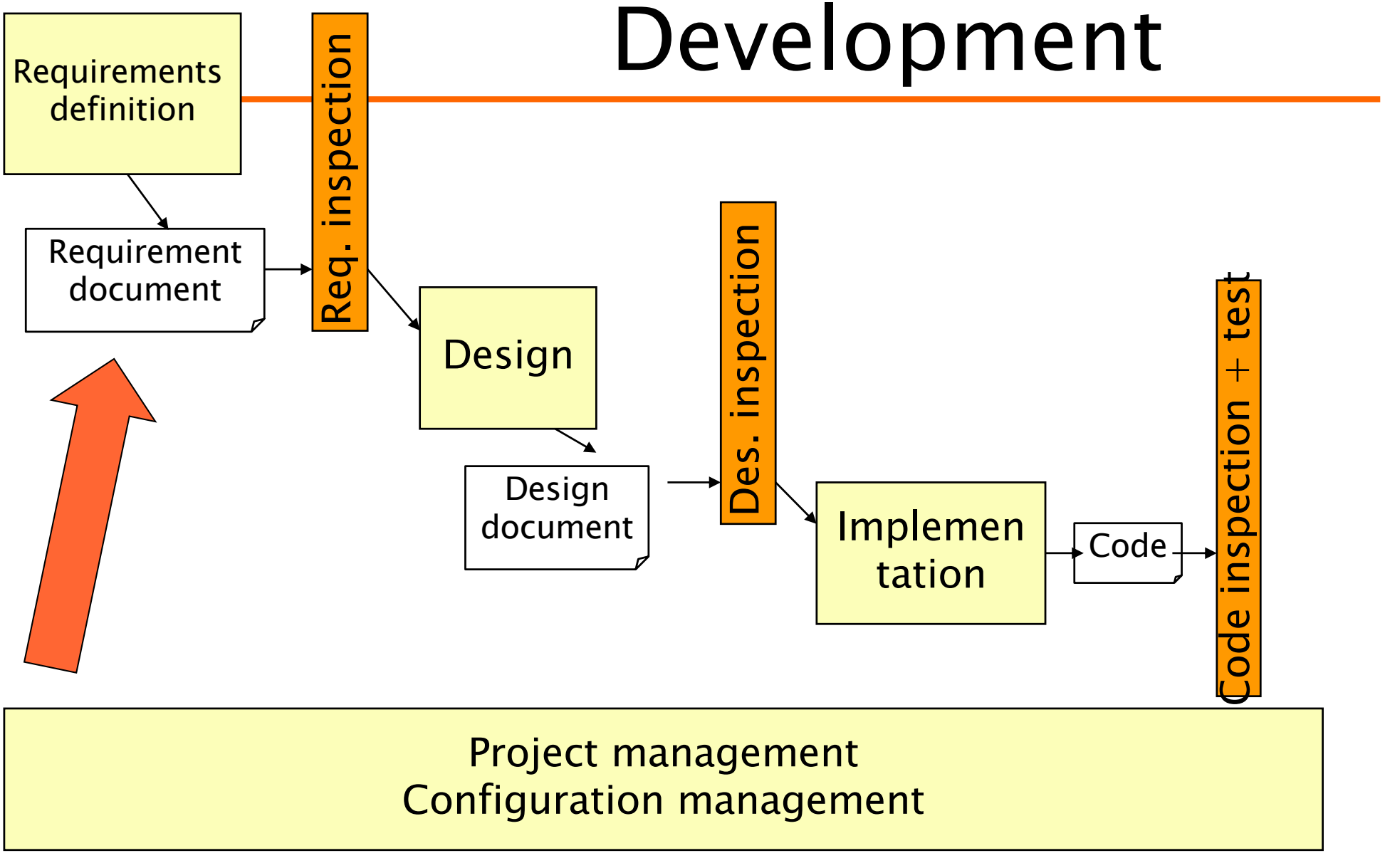
Outline

- Stakeholders
- Context diagram and interfaces
- Types of requirements
- Numbering requirements
- Scenarios, sequence diagrams
- Use cases
- Glossary
- Class diagrams

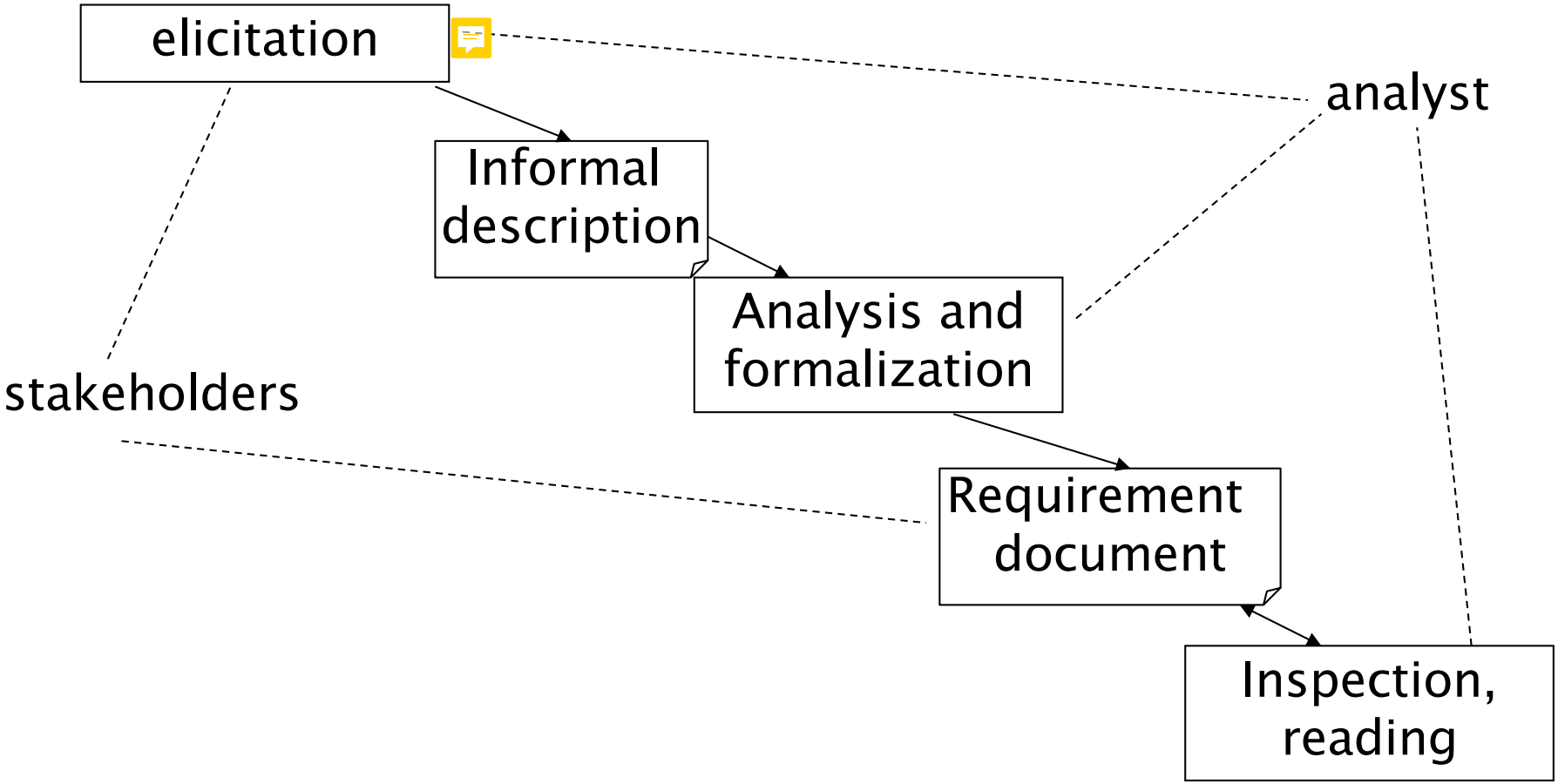
Main Phases


Development
deployment
Operation
retirement
Maintenance
t

Development



Requirement engineering



Requirement

- Description of a system / service and of constraints on it
  - Different levels of abstraction
  - Different levels of formality
  - Functional – not functional

Requirements – informal

- A POS (Point-Of-Sale) system is a computer system typically used to manage the sales in retail stores. It includes hardware components such as a computer, a bar code scanner, a printer and also software to manage the operation of the store.
- The most basic function of a POS system is to handle sales.

Requirements – informal

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
  - ‘manage sales’ and ‘handle sales’ mean the same thing? What is a sale?

Completeness and consistency

- In principle, requirements should be both complete and consistent.
  - Complete
    - They should include descriptions of all facilities required.
  - Consistent
    - There should be no conflicts or contradictions in the descriptions of the system facilities.
  - In practice, it is impossible to produce a complete and consistent requirements document.

Requirements – defects

- Omission/ incompleteness
- Incorrect Fact
- Inconsistency/contradiction
- Ambiguity
- Extraneous Information
  - Overspecification (design)
- Redundancy

Requirement techniques

- Stakeholders
- Context diagram
- Requirement numbering and type
- Glossary
- Use case diagram
- Scenario, sequence diagram
- System diagram

Requirement doc structure

1 Overall description
2 Stakeholders
3 Context diagram and interfaces
4 Requirements
   - Functional
   - Non functional
   - Domain
5 Use case diagram
6 Scenarios
7 Glossary
8 System design

Req document and techniques

- The requirements document provides a structure
- Within the structure different techniques can be used
- The structure may change, order of parts is less important than precise description of parts

Req document vs. techniques

Overall description	Text
Stakeholders	Text
Context diagram	UML UCD
Interfaces	Text, PDL, XML, screenshots
Requirements	Type, numbering
Use cases	UML UCD
Scenarios	Tables, text
Glossary	Text, UML CD
System design	UML CD

Other requirement structures

- http://readyset.tigris.org

- IEEE Std 830 1994
  - Introduction.
  - General description.
  - Specific requirements.
  - Appendices.
  - Index.

Techniques

Stakeholder

- Role or person with an interest (stake) in the system to be built

- User
  - Uses the system
  - Can include different user profiles
- Buyer
  - Pays for the system
- Administrator
- Analyst
  - Expert in requirement engineering, and or in domain
- Developer

Stakeholders – example

- Point Of Sale (POS) in a supermarket

- User
  - Cashier at POS (profile 1)
  - Supervisor, inspector (profile 2)
  - Customer at POS (indirectly through cashier)
- Administrator
  - POS application administrator (profile 3)
  - IT administrator (profile 4)
    - Manages all applications in the supermarket
  - Security manager (profile 5)
    - Responsible for security issues
  - DB administrator (profile 6)
    - Manages DBMSs on which applications are based
- Buyer
  - CEO and/or CTO or CIO of supermarket

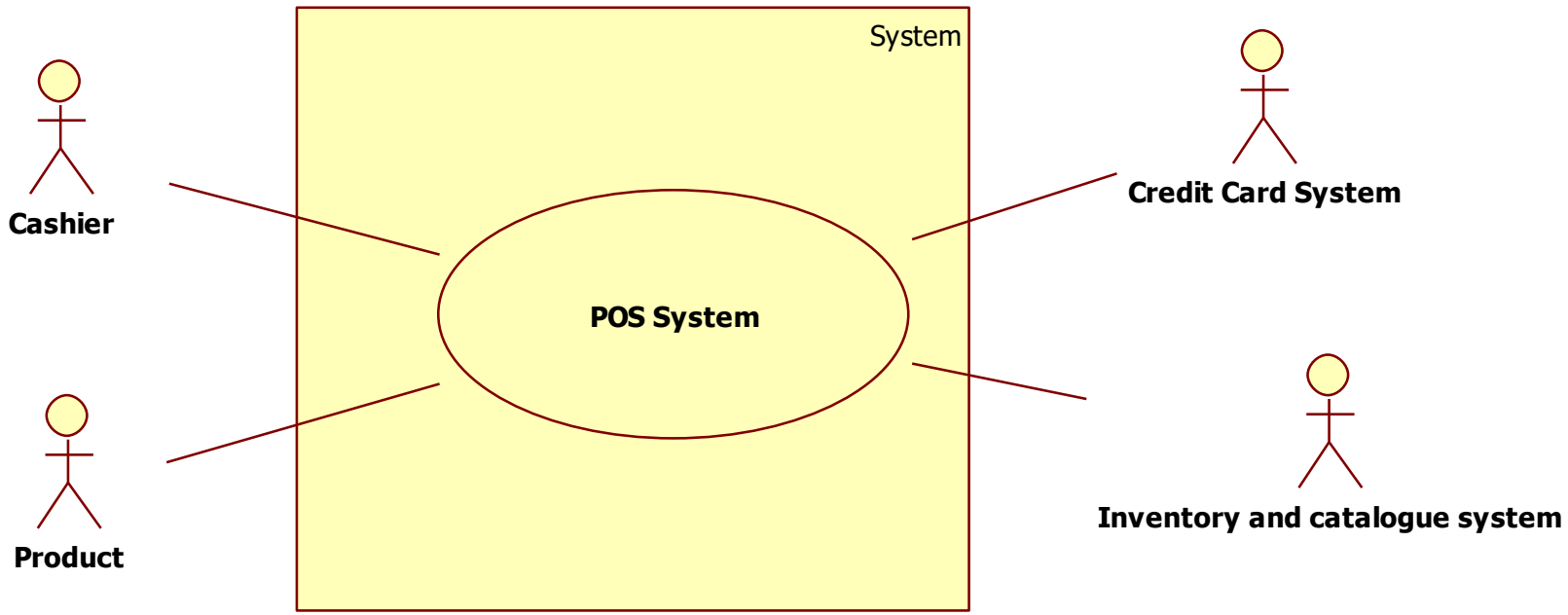
Stakeholders

- Listing the relevant stakeholders is essential to consider relevant points of view (and therefore relevant requirements) for a system

Context diagram

- Defines what is **inside** the system to be developed, what is **outside**
  - Entity outside = actor
  - {actor} ⊂ {stakeholder}
  - Other systems/subsystems/applications
  - Human users
- Defines **interfaces** between inside and outside

Context diagram



Interfaces

- With cashier
  - Physical level: Screen, keyboard
  - Logical: GUI (to be described)
- With product
  - Physical level: laser beam (bar code reader)
  - Logical level: bar code
- With credit card system
  - Physical level: internet connection
  - Logical: web services (functions to be described, data exchanged, SOAP + XML)

Interface specification

- Three types of interface may have to be defined
  - User interfaces, GUIs
  - Procedural interfaces;
  - Data exchanged;
- Formal notations are an effective technique for interface specification.

PDL interface description

```
interface PrintServer {

// defines an abstract printer server
// requires:          interface Printer, interface PrintDoc
// provides: initialize, print, displayPrintQueue, cancelPrintJob, switchPrinter

        void initialize ( Printer p ) ;
        void print ( Printer p, PrintDoc d ) ;
        void displayPrintQueue ( Printer p ) ;
        void cancelPrintJob (Printer p, PrintDoc d) ;
        void switchPrinter (Printer p1, Printer p2, PrintDoc d) ;
} //PrintServer
```

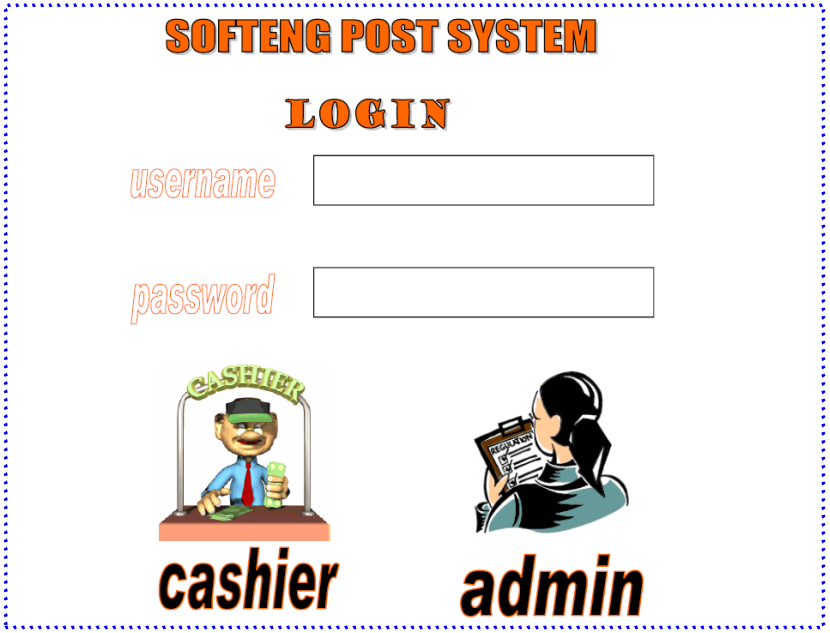
Data interface (XML)

```
<food>
<name>Belgian Waffles</name>
<price>$5.95</price>
<description>
two of our famous Belgian Waffles with plenty of real maple syrup
</description>
<calories>650</calories>
</food>

<food>
<name>Strawberry Belgian Waffles</name>
<price>$7.95</price>
<description>
light Belgian waffles covered with strawberries and whipped cream
</description>
<calories>900</calories>
</food>
```

GUI interface

- Sketch of interface, typically built with GUI builder



Context diagram and interfaces

- Are essential to agree on
  - What is the scope of the system (and therefore which requirements should offer / wich functions should use from outside)
- The scope changes radically cost and time to develop
  - Ex. GIS, maps are developed and managed internally, or uses Google maps?

Requirement types

- Functional
  - Description of services / behaviors provided by the system
  - Application vs. domain
- Non functional
  - Constraints on the services
  - Application vs. domain
  - (Domain
    - From the domain (= set of related applications, ex banking, telecom))

Functional

Requirement ID	Description
F1	Handle sale transaction
F2	Start sale transaction
F3	End sale transaction
F4	Log in
F5	Log out
F6	Read bar code

Naming the FR

- Managememt
  - FR is ready or not?
  - Is released or not
  - Paid or not
- Testing level
  - Tested or not?

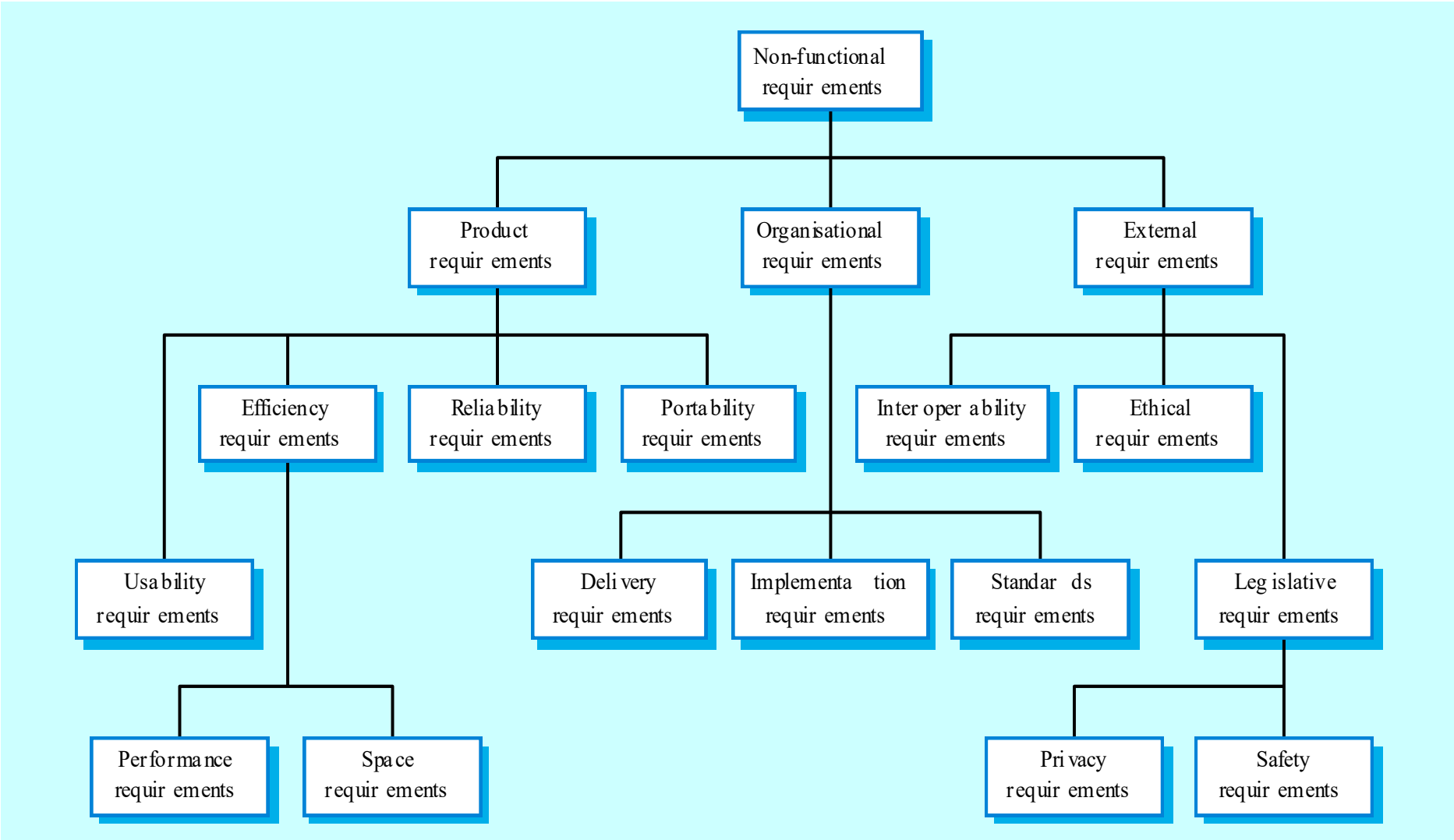
Non Functional

- Efficiency
- Usability
- Reliability
- …

ISO 9126 / ISO 25010

- Defines 6 properties of software systems
  - 5 non functional
    - Functionality
    - Reliability
    - Usability
    - Efficiency
    - Maintainability
    - Portability

Non-functional requirements



Non-functional reqs

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Perception of waiting times

- <0.1 sec – not percepted
- >1 sec – annoying

Non Functional

Requirement ID	Description
NF1(efficiency)	Function F1.1 less than 1msec
NF2 (efficiency)	Each function less than ½ sec
Domain1	Currency is Euro – VAT is computed as ..

Non-functional requirements

- Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

NF Req must be measurable

- **Not measurable**
  - The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised.
- **Measurable**
  - Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

Measures for NF reqs

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	M Bytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

- NF requirements such as
  - «The system should be easy to use»
  - «The system should be maintainable»
  - «The system should be portable»

- Are not testable, and therefore useless

Requirements interaction

- Conflicts between different non-functional requirements are common in complex systems.
- Spacecraft system
  - To minimise weight, the number of separate chips in the system should be minimised.
  - To minimise power consumption, lower power chips should be used.
  - However, using low power chips may mean that more chips have to be used. Which is the most critical requirement?

Domain requirements

- Derived from the application domain and describe system characteristics and features that reflect the domain.
- Domain requirements can be new functional requirements, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

Train protection system

- The deceleration of the train shall be computed as:
  - $D_{train} = D_{control} + D_{gradient}$

  where $D_{gradient}$ is $9.81 ms^2$ * compensated gradient/alpha and where the values of $9.81 ms^2$ /alpha are known for different types of train.

Domain req. problems

- Understandability
  - Requirements are expressed in the language of the application domain;
  - This is often not understood by software engineers developing the system.
- Implicitness
  - Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

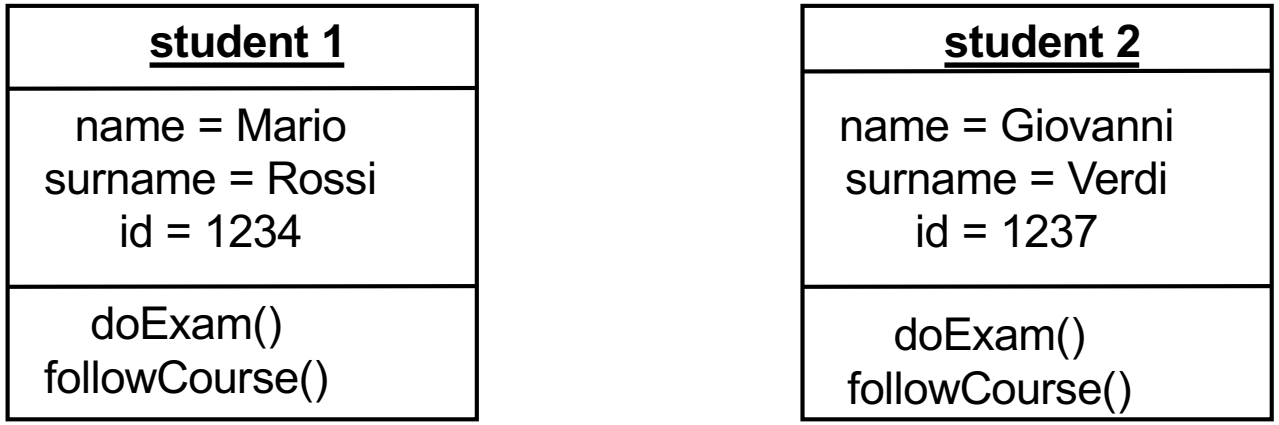
Glossary

- Sale = commercial transaction between customer and retailer, where customer buys a number of products from the retailer
- Product = ..

Class / object models

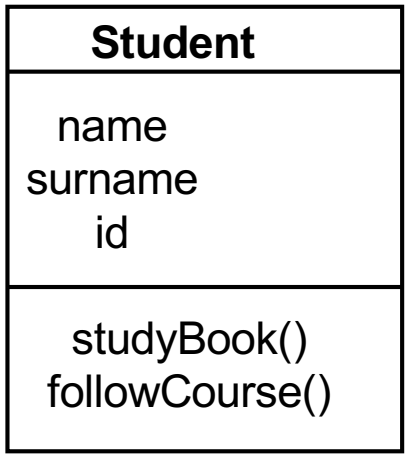
Object

- Model of entity (physical or inside software system)
  - ex.: student, exam, stack, window
- characterized by
  - identity
  - attributes (or data or properties)
  - operations it can perform (behaviour)
  - messages it can receive
- graphic representation: rectangle


student 1
name = Mario
surname = Rossi
id = 1234
doExam()
followCourse()
student 2
name = Giovanni
surname = Verdi
id = 1237
doExam()
followCourse()

Class

- Descriptor of objects with similar properties

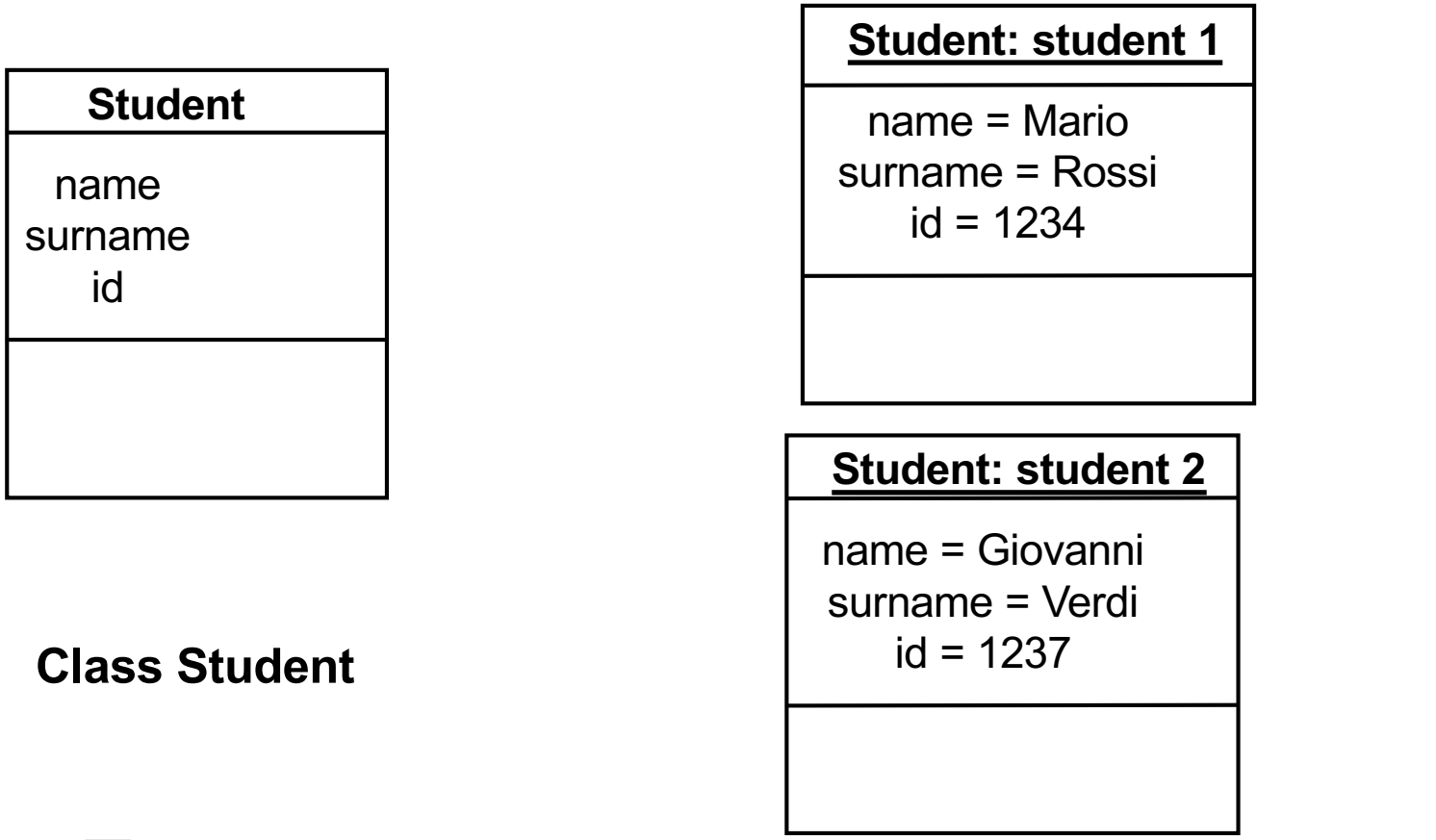


Class – cont.

- attribute
  - the name of an attribute is the same for all objects and can be described in the class
  - the value of an attribute may be different on each object and cannot described in the class
- Operation / behaviour
  - is the same for all objects and can be described in the class
  - will be applied to different object (possibly with different results)

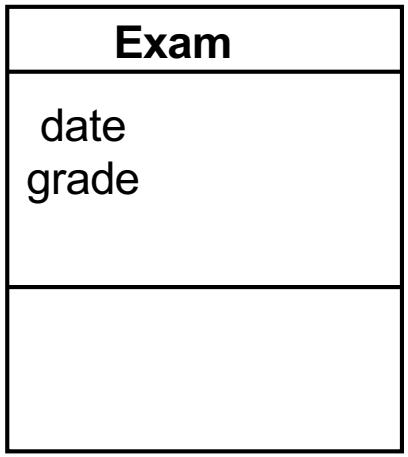
Class and object

- object is instance of a class

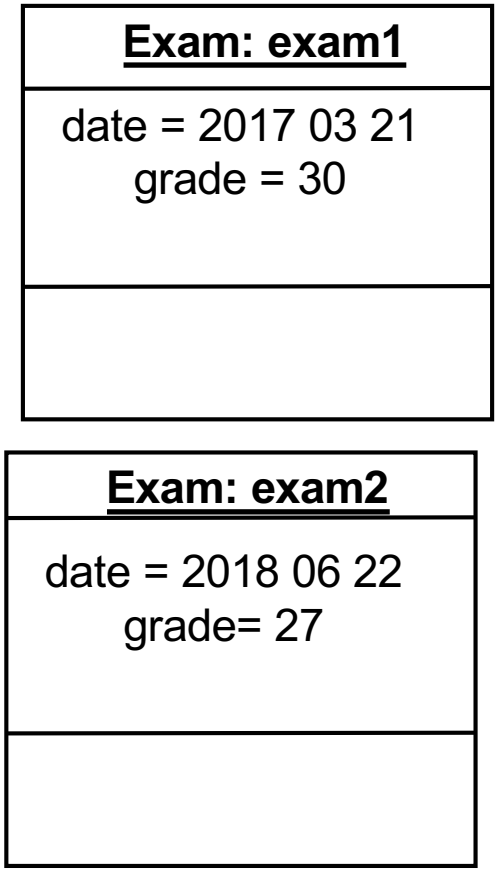


Class and object

- Exam: the result of an assessment of a student on a certain course



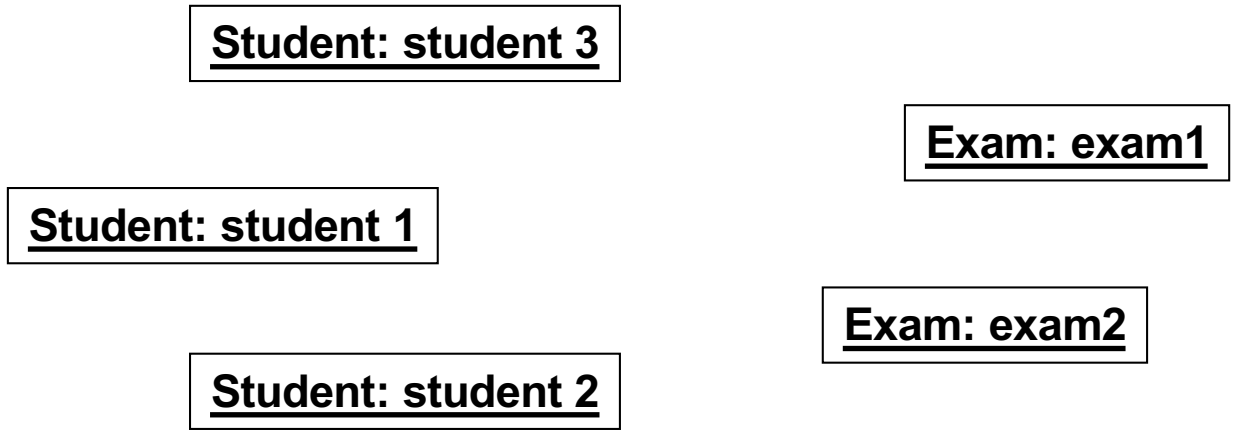
Class Exam



objects (instances) of class Exam

Object diagram

- Models objects of interest in a specific case



  - Remark: above is a reduced notation for object/class
  - Remark: links are key part of diagram, see next slides

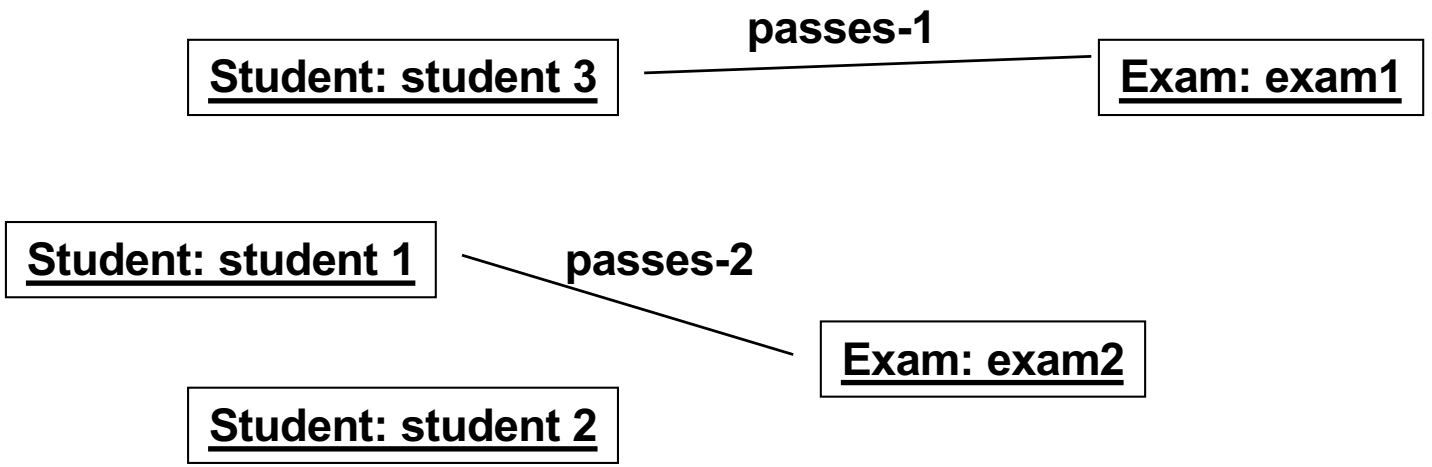
Class diagram

- Models classes of interest in a specific case

  - Remark: relationships are key part of this diagram, see next slides

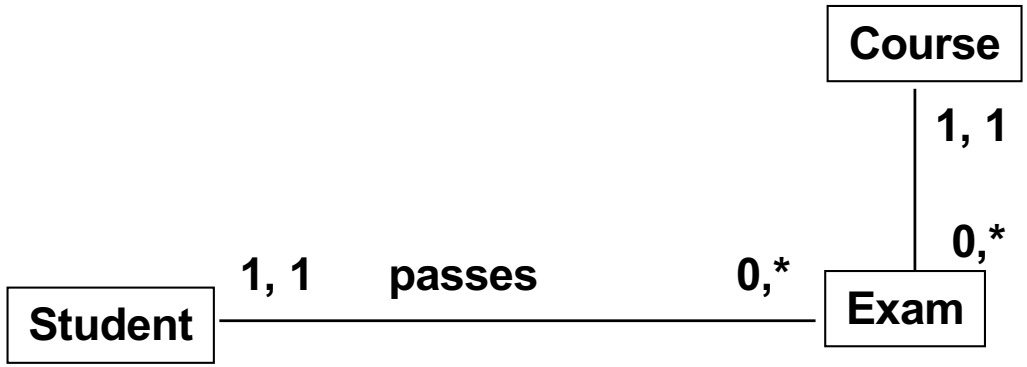
Link

- Model of association between objects

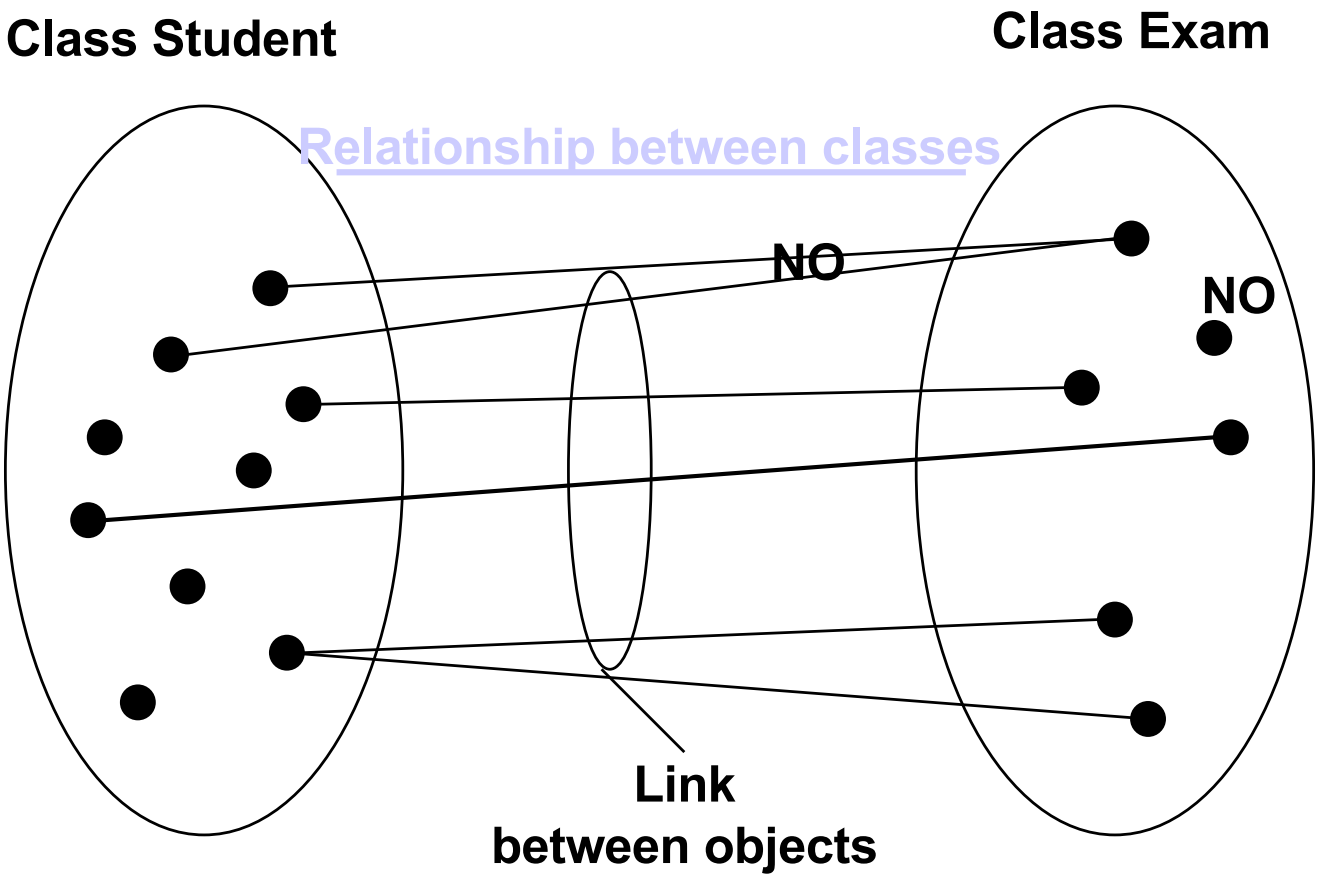


Relationship

- Descriptor of links with similar properties



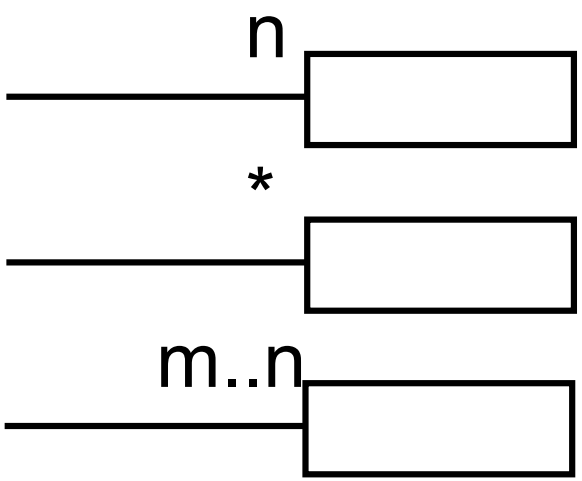
Relationships



Multiplicity

- Constraint on max / min number of links that can exit from an object

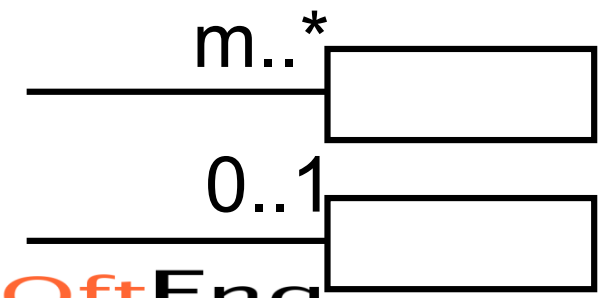
Multiplicity



Exactly n

Zero or more

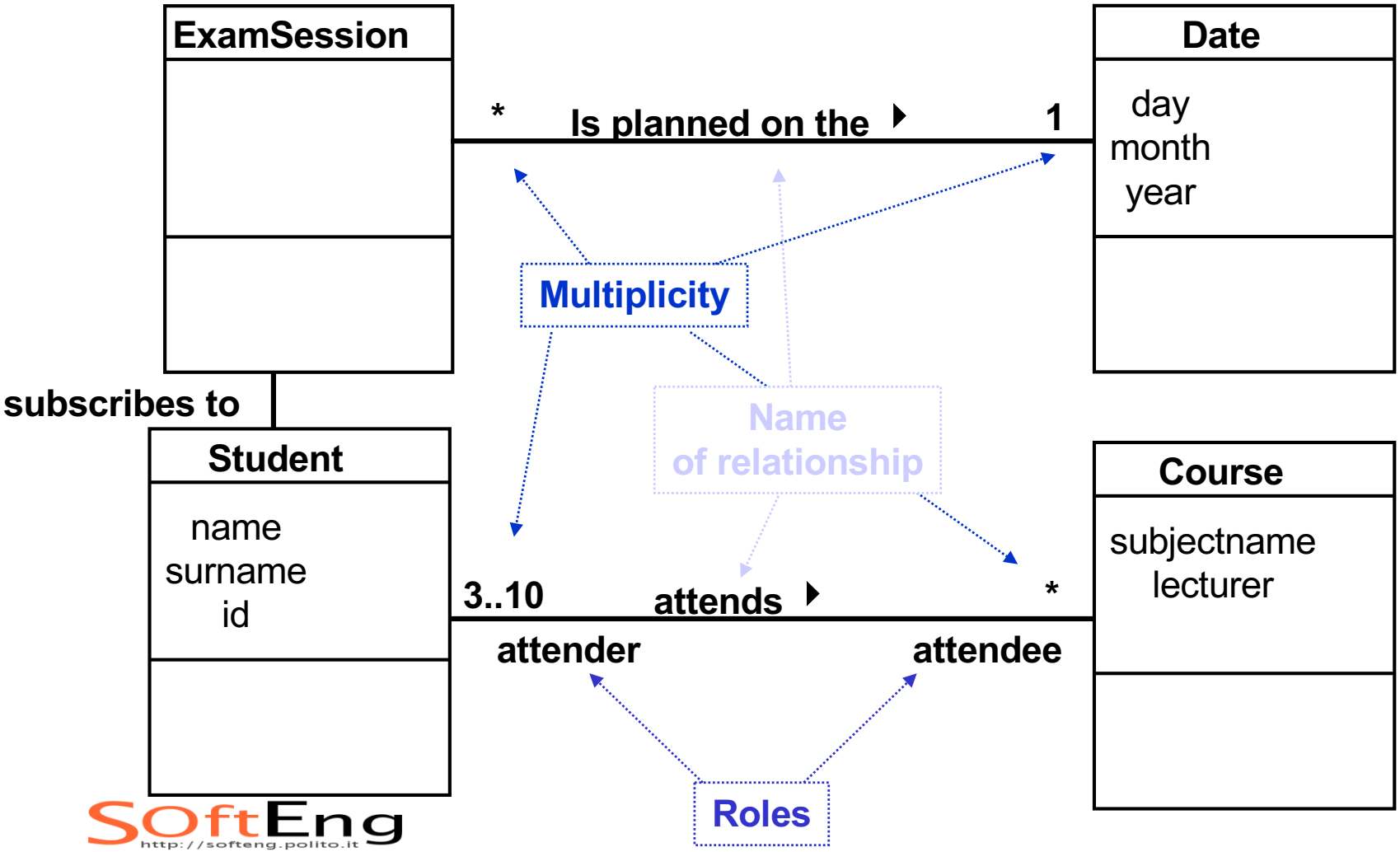
between m and n (m,n included)



from m up

Zero or one (optional)

Relationships



Relationships

- A relationship is always bidirectional
  - A student passes an exam
  - An exam is passed by a student
  - At implementation level (programming language or database) the relationship may or may not be implemented in both directions

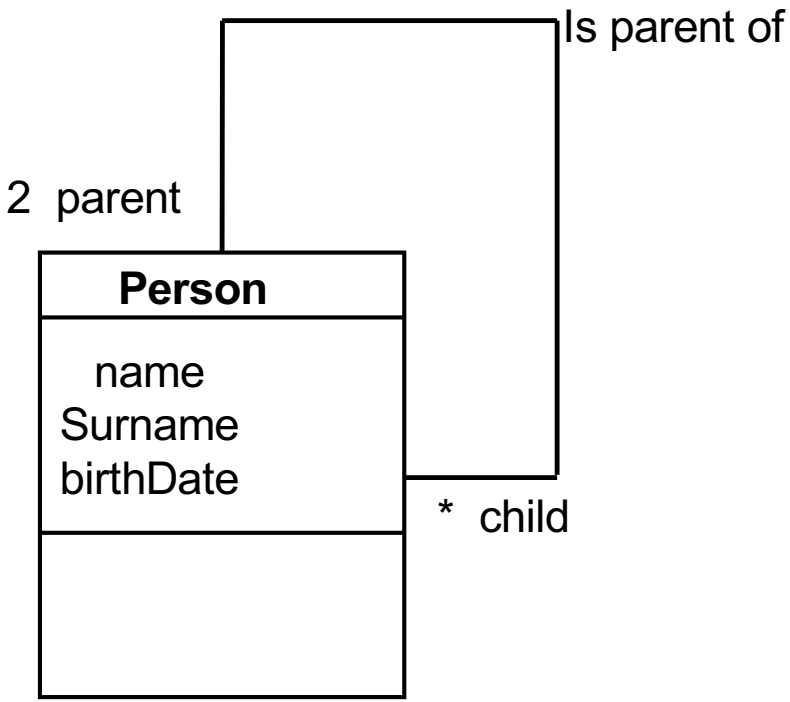
Other constraints

- Constraints (especially the ones involving more classes and relationships) can be expressed textually using OCL (Object Constraint Language)

Invariant: Exam.grade >= 18 && Exam.grade<= 33

Invariant: ExamSession.isPlannedOn.Date != weekEndDay

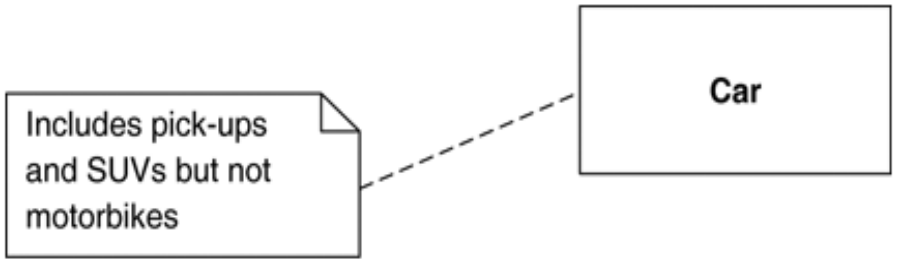
Reflexive relationships



Person Invariant: this.parent->forall(p| p.birthDate <this.birthDate)

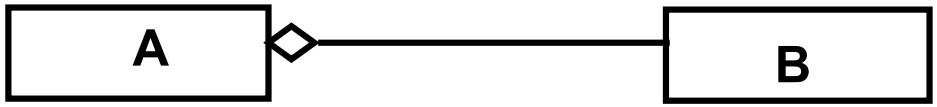
Notes and Comments

- **A note is used as a comment on one or more diagram elements**

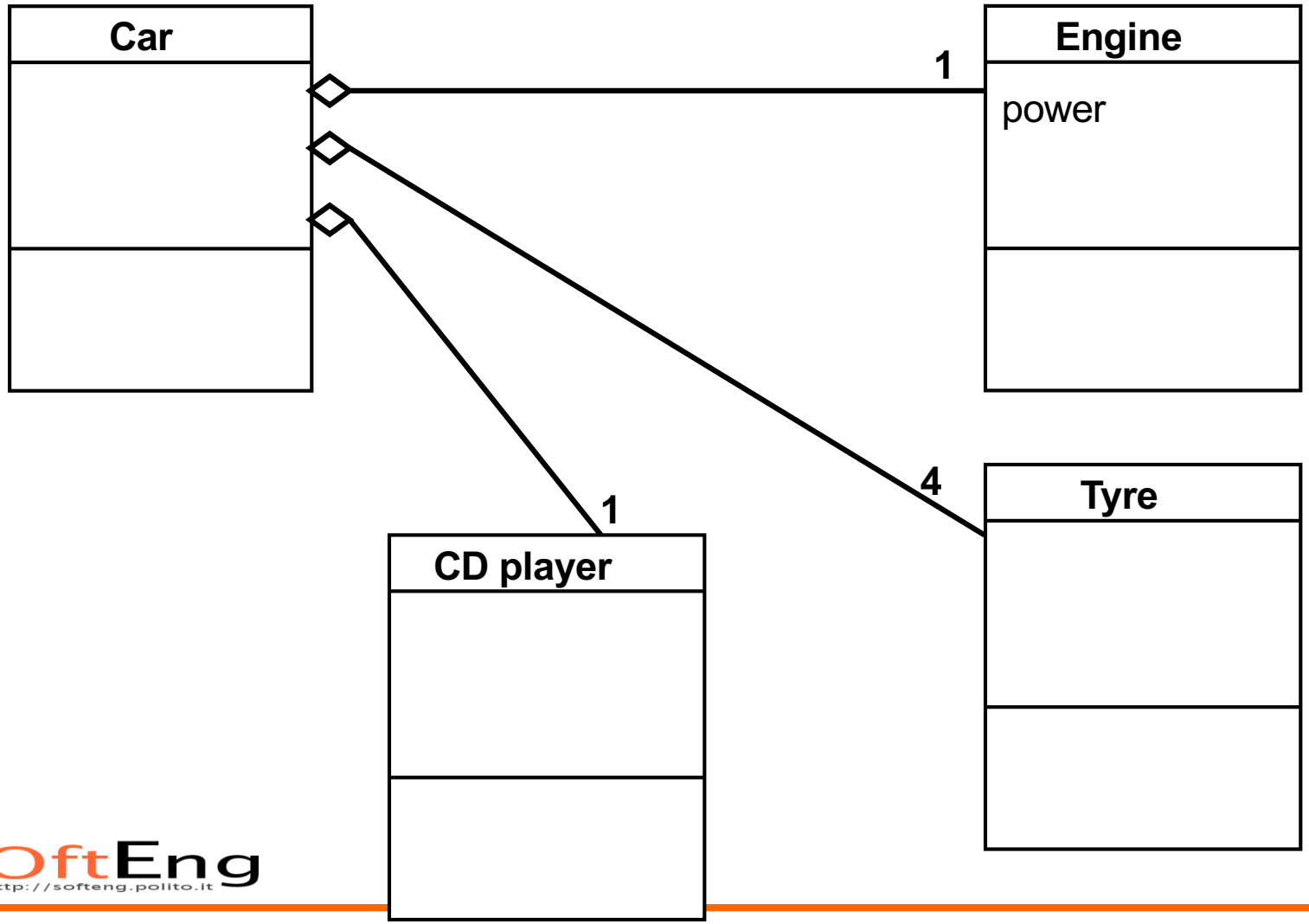


Aggregation

- B *is-part-of* A means that objects described by class B can be attributes of objects described by A



Example



Specialization

- or Generalization, or is-a

- A *specializes* B means that objects described by A have the same properties (attributes, operations) of objects described by B
- Objects described by A can have additional properties

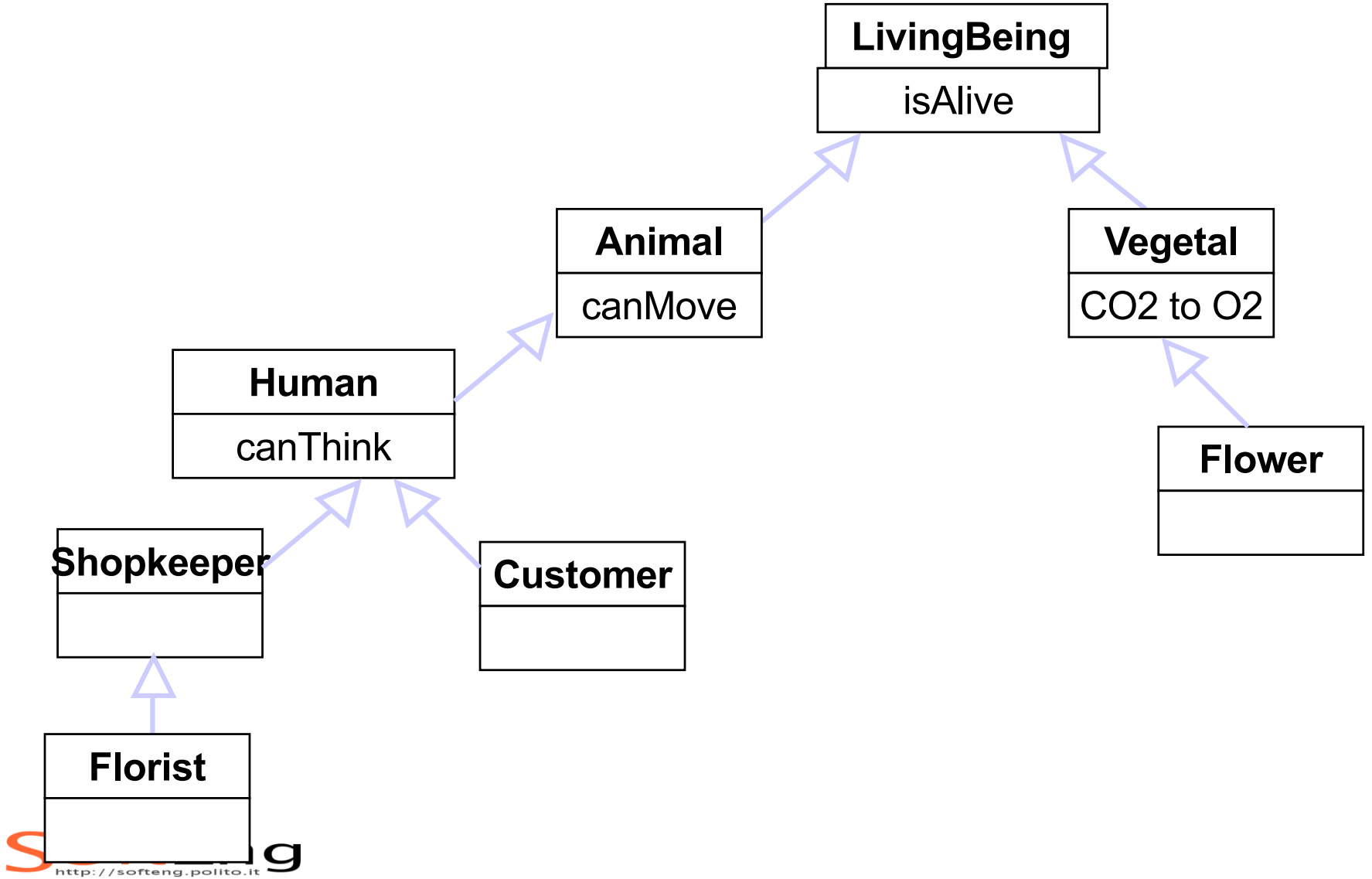
Subclass superclass

- Subclass = specialized class
- Superclass = generalization class

Inheritance

- mechanism associated to specialization/generalization relationship
- properties defined by B are inherited by A
  - A does not need to repeat these properties
  - Human
    - canThink (own property)
    - canMove (inherited from Animal)
    - IsAlive (inherited from LivingBeing)

Example


LivingBeing
isAlive
Animal
canMove
Vegetal
CO2 to O2
Human
canThink
Flower
Shopkeeper
Customer
Florist

Role vs. Specialization

- An object (instance of a class) can change role
  - P1 : instance of Player
  - P1 has role captain, then P2 becomes captain
- An object (instance of a class) cannot change class
  - P1 remains instance of player forever (cannot become instance of another class)

In short

- Object diagram (models)
  - object
  - link
- Class diagram (descriptors)
  - class
  - relationship
    - aggregation
    - specialization
  - multiplicity
- to model structural information
  - structural viewpoint

DO in class diagram

- Decide goal of diagram

- Goals can be:
  - Model of concepts (glossary)
  - Model of system (hw + sw) == system design
  - Model of software classes (softwaredesign)

DO in class diagram / conceptual model

- What classes to use in model?
  - Physical – student, person, airplane, classroom
  - Descriptors – aircraft type, product type, car type
  - Organizational entities – airline, university, department
  - Time
    - Events: landing, take off, subscription to course, exam
    - Time ranges: flight, academic year, course, meeting, lesson
  - Exchange of money / service / good
    - Sale, purchase, payment, subscription to course, shipping

DO in class diagram / conceptual model

- What classes to use in model?
  - Geographical entities
    - City, region, country, land ..
  - Reports, summaries
    - Weather report, weather forecast, student enrollment report.. , bank account statement, wish list

DO NOT in class diagrams

- Use plurals for classes
  - Classroom yes, no classroomS
- Use transient (dynamic) relationships
  - (they will be modeled in scenarios, sequence diagrams)
- Checkout loops (cycles in graph)
- Forget multiplicities
- Forget roles / association classes, when needed
- Use class as an attribute
- Use attribute that represents many objects

DO NOT in class diagrams

- Repeat as an attribute of a class a relationship starting from the class
- Confound system design, software design, glossary
  - DO decide goal of diagram

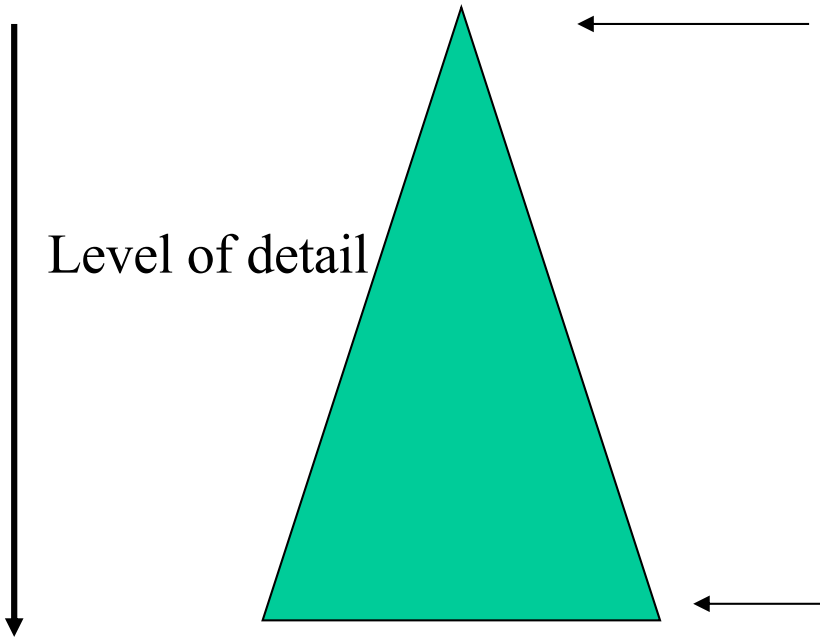
Support from OO prog. languages

- object, class
  - supported
- relationships
  - aggregation
    - supported partially
  - specialization
    - supported

Use of class diagrams

- Class diagrams are just a notation
- can be used in different documents with different goals
  - user requirements
  - developer requirements
  - system design
  - (unit design)

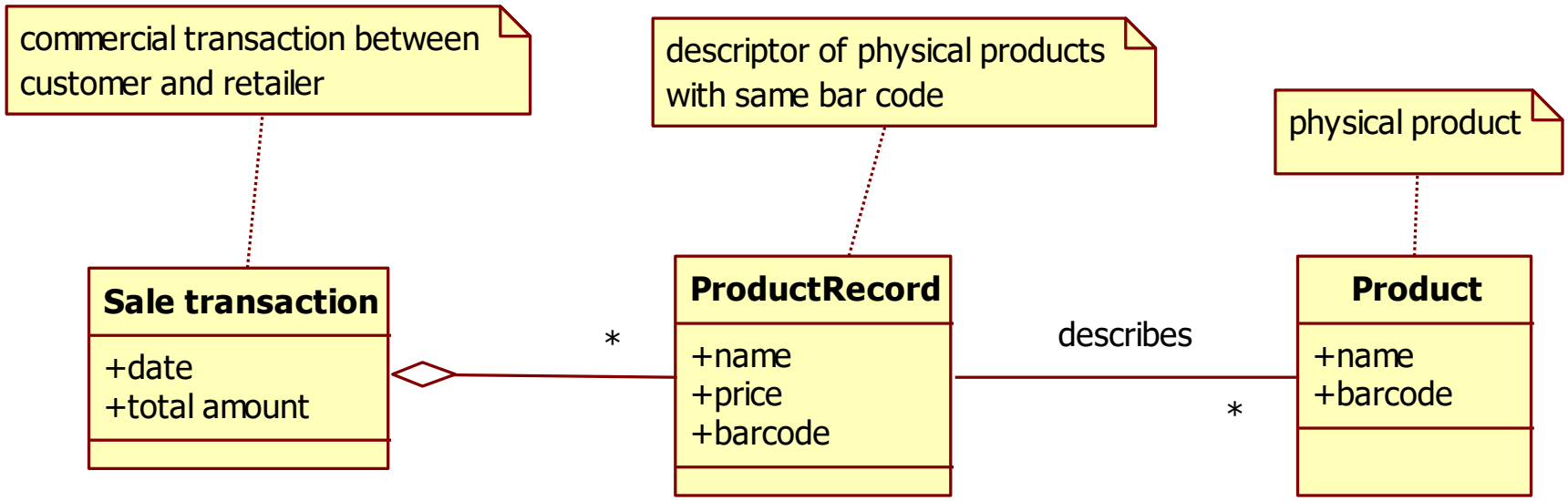
Use of class diagrams



Conceptual model
class diagram with
few domain level classes

Source code:
class diagram reverse engineered
from code

Glossary with Class diagram



What classes in UML for glossary?

- For physical objects
  - Ex Pilot, Aircraft, Airline, Airport
- For legal / organizational entities
  - Company, airline, university, department
- Descriptors
  - Ex Aircraft type, Pilot Qualification
- Time (events)
  - Departure of aircraft
- Commercial transaction
  - Event + exchange of money / good / service
- Time (intervals)
  - Flight, Internship

Scenario

- Sequence of steps (events) that describe a typical interaction with the system

Step	Description
1	Start sales transaction
2	Read bar code
3	Retrieve name and price given barcode
	Repeat 2 and 3 for all products
4	Compute total
5	Manage payment cash
6	Deduce stock amount of product
7	Print receipt
8	Close transaction

Scenario / pre post conditions

- Precondition
  - Condition to be satisfied before starting the scenario
- Postcondition
  - Condition satisfied at end of the scenario

Use case

- Set of scenarios with common user goal

- Ex: use case: Handle sales
  - Scenario1: sell n products
  - Scenario2: sell n products, abort sale because customer has no money
  - Scenario 3: sell n products, customer changes one of products

- Ex other use cases
  - Handle coupons
  - Handle users

Use cases

Use Cases

- Semi-formal notation
- Study of the application domain
- Identification of boundaries and interactions among the system and the external world
- Useful to
  - Oblige the analyst to state well-defined boundaries between system and external world
  - Organize system functions into elements (use cases) on which attention is focused
  - Supply a first basis for the specification of system structure from the user perspective

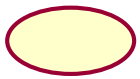
Use Case Diagram

- Provide a more functional view of a software system
  - functions, actors
  - boundary
- Readable by customer/user
- Usually defined before class diagrams
- Diagram composed of actors, use cases, relationships

Elements of a Use Case

Actor

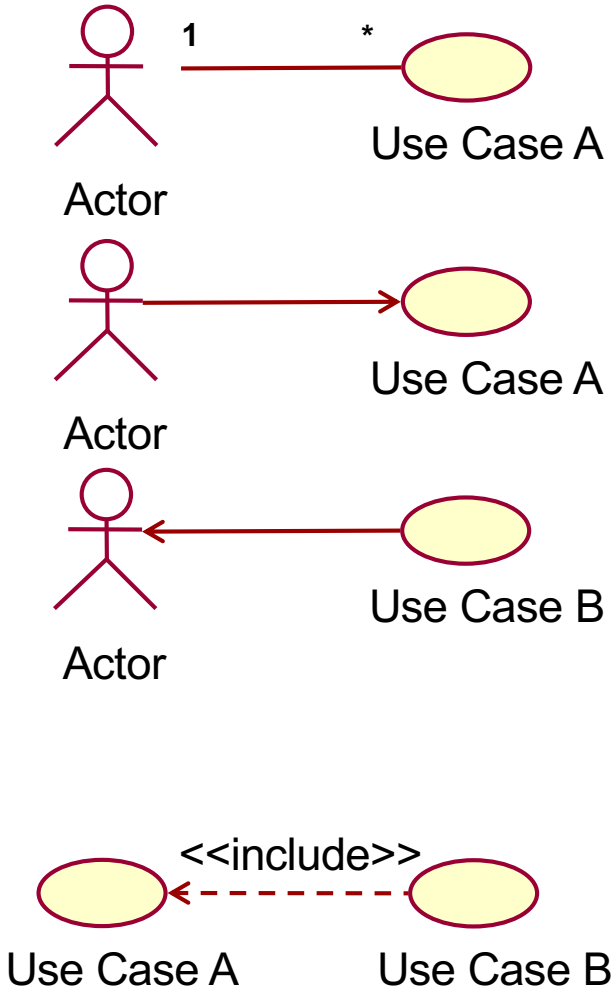
- Someone (user) or something (external system, hardware) that
  - Exchanges information with the system
  - Supplies input to the system, or receives output from the system



Use Case

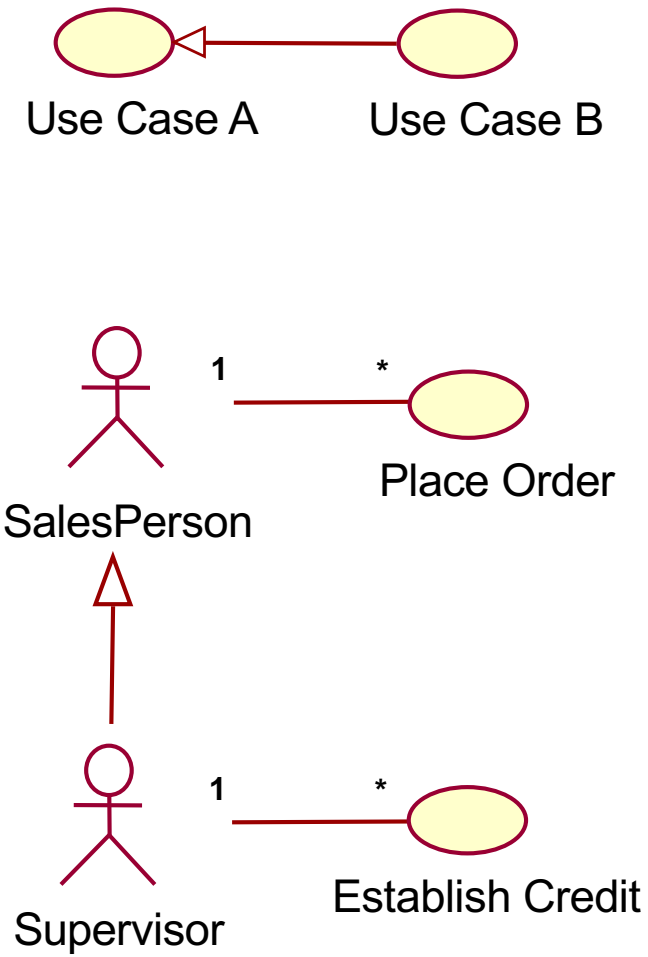
- A functional unit (functionality) part of the system

Relationships



- Association models:
  - Which actors participate in a use case
  - Where execution starts
  - Adornments (e.g. multiplicity, direction) allowed
  - Actor1 participates in Use CaseA and is the trigger of the use case
  - Actor2 participates in UseCaseB and UseCaseB is the trigger
- Include
  - Models that functionality A is used in the context of functionality B (one is a phase of the other)

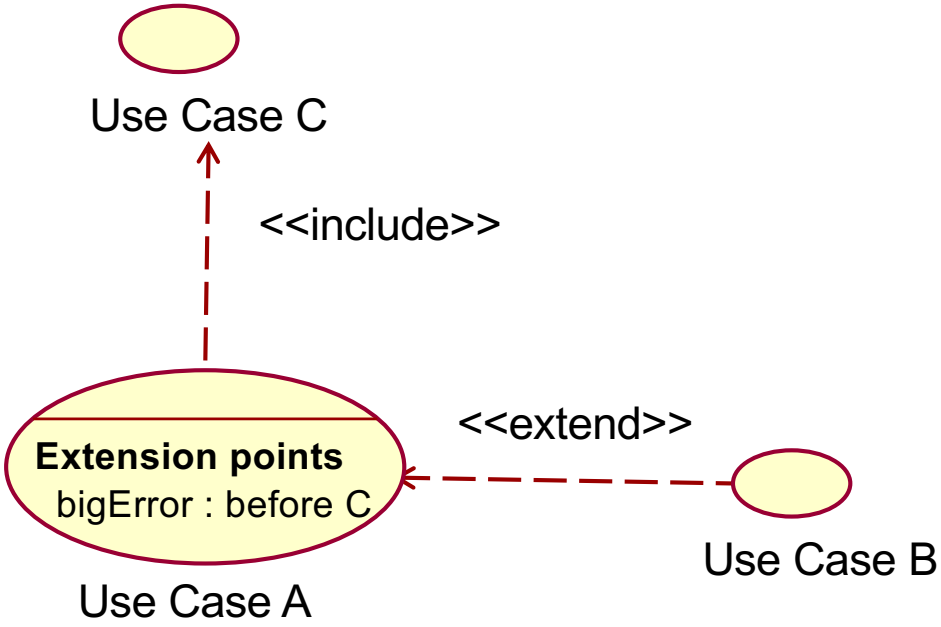
Relationships: generalization



- Generalization
  - Defines functionality B as a specialization of functionality A (e.g. a special case)
- Generalization
  - A generalization from an actor B to an actor A indicates that an instance of B can communicate with the same kinds of use-case instances as an instance of A

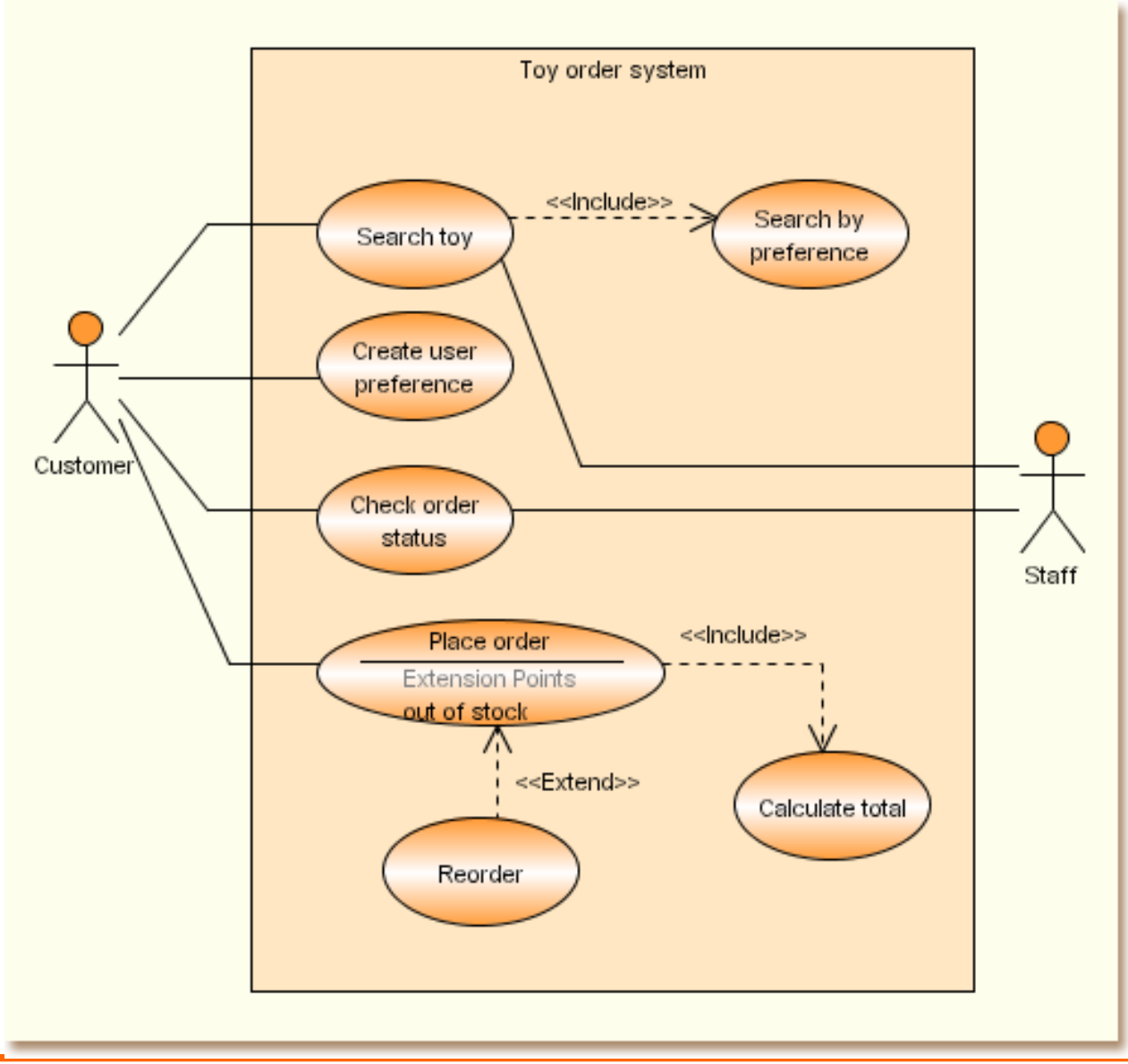
SOftEng
http://softeng.polito.it

Relationships: extension

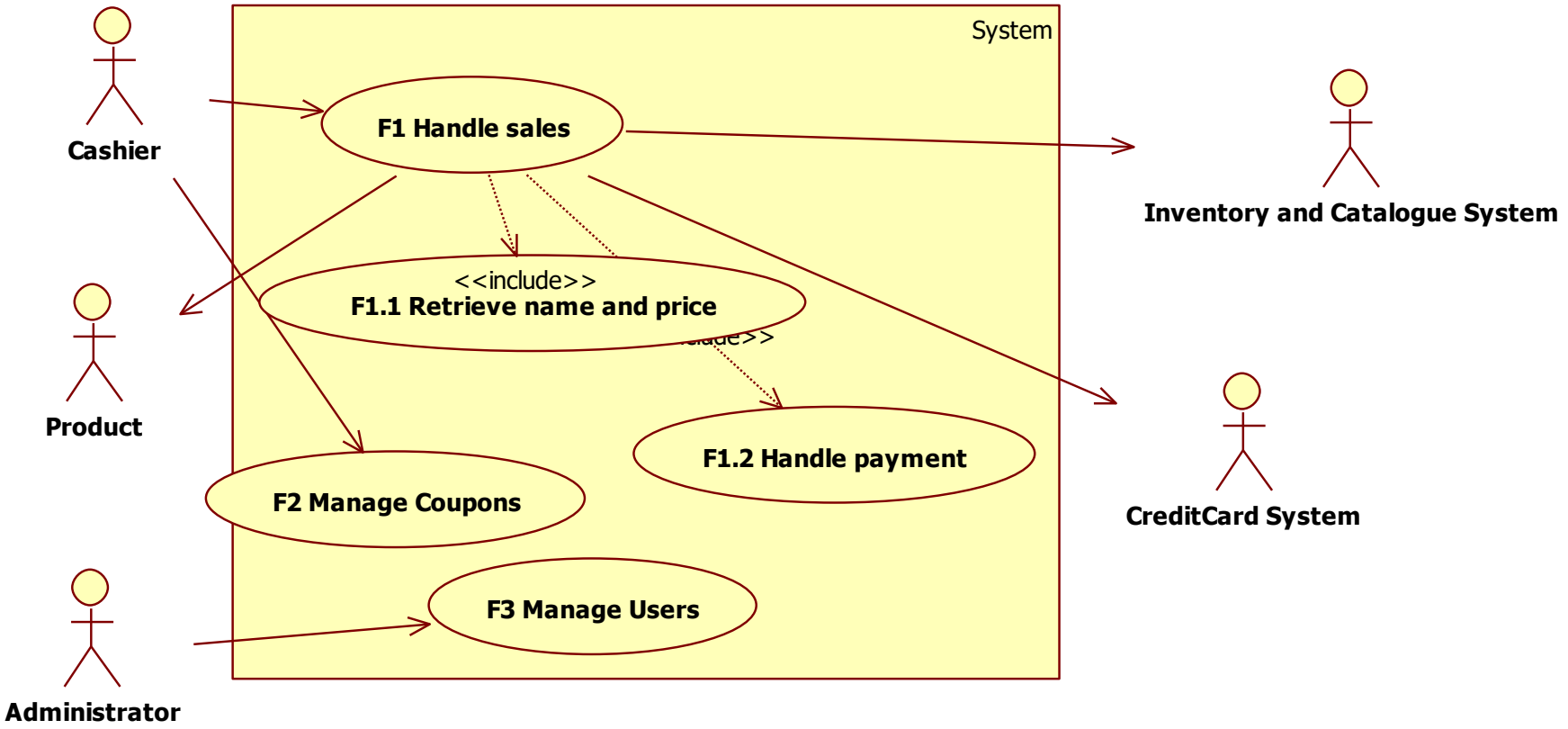


- Extension
  - An extend relationship from use case A to use case B indicates that an instance of use case B may be augmented by the behavior specified by A
  - The behavior is inserted at the location defined by the extension point (name : where) in B, which is referenced by the extend relationship

Use case – Example



Use case diagram



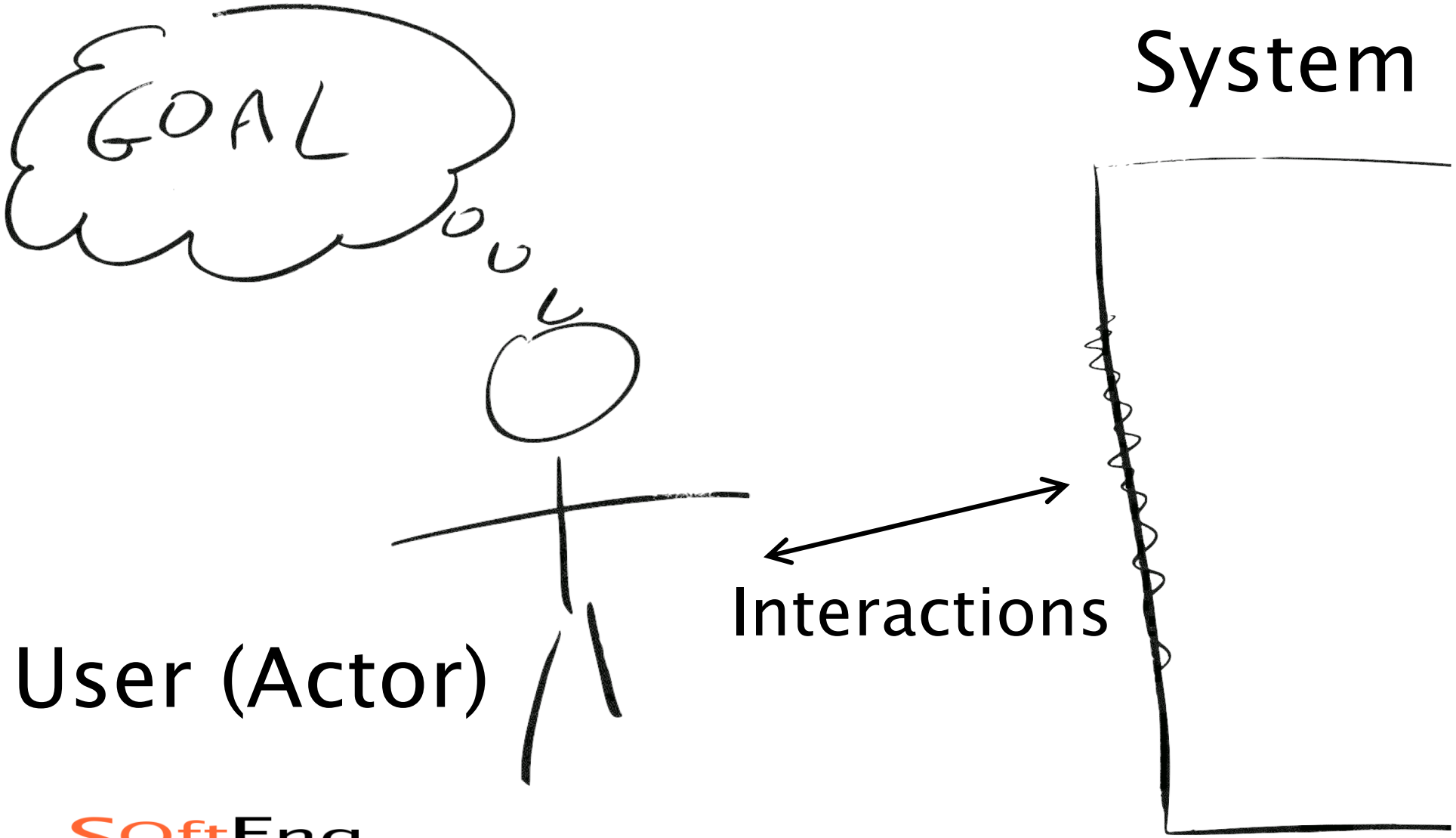
Use case

- Captures a contract between the actors of a system about its behavior.
- Describes the system's behavior under various conditions as it responds to a request
  - from the *primary actor.*
- The primary actor initiates an interaction with the system to accomplish some goal.
- The system responds, protecting the interests of all the stakeholders.

Use case

- A scenario is a sequence of steps describing an interaction between a user and a system
- A use case is a **set of scenarios** tied together by a **common *user* goal.**

Key elements



Key elements

- The actor involved
  - type of user that interacts with the system
- The system being used
  - treated as a black-box
- The functional goal that the actor achieves using the system
  - the reason for using the system

Actors, stakeholders

- Actor <= stakeholders
- External to system
- Actors can be
  - Humans
  - Other machines / systems
- Actors can be
  - Primary: start the interaction with the system
  - Secondary: are passive wrt the system

Goals

- The use case cares only what is the relationship of the actor to the system
- The goal must be of value to the (primary) actor:
  - “Enter PIN code” is not
  - “Withdraw cash” is

Goal

As a *<actor type>*

I want *<to do something>*

So that *<some value is created>*

Goal

bank customer

As a *<actor type>*

to perform a withdrawal

I want *<to do something>*

I get some cash for me

So that *<some value is created>*

Use case briefs

- Summary consisting of 2-6 sentences
  - What is going on
  - Most significant activities

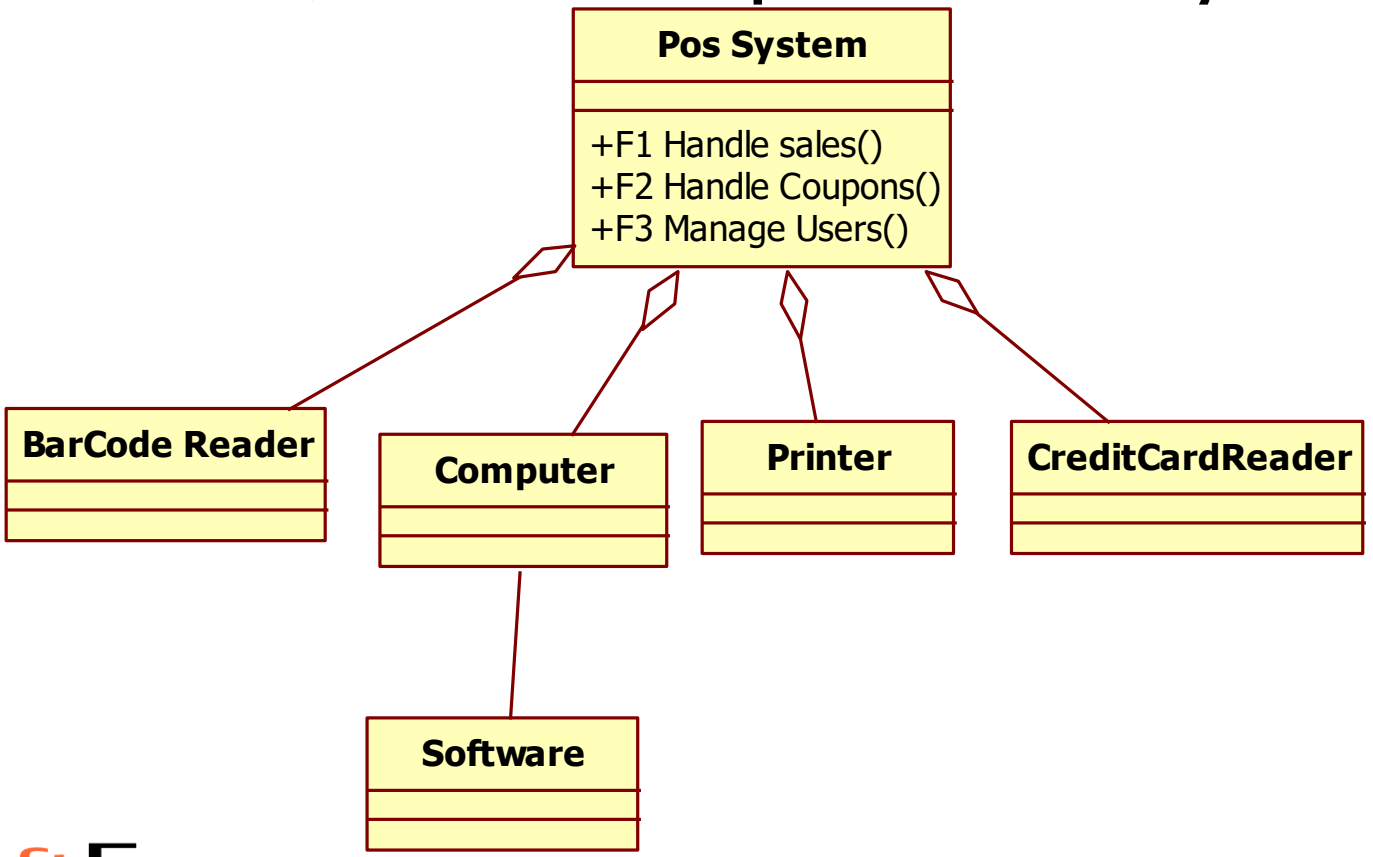
Actor	Goal	Brief
Production Staff	Modify the administrative area lattice	Production staff add admin area metadata (administrative hierarchy, currency, language code, street types, etc.) to reference database and contact info for source data is cataloged. This is a special case of updating reference data.
Production Staff	Prepare digital cartographic source data	Production staff convert external digital data to a standard format, validate and correct it in preparation for merging with an operational database. The data is catalogued and stored in a digital source library.
Production & Field staff	Commit update transactions of a shared checkout to an operational database	Staff apply accumulated update transactions to an operational database. Non-conflicting transactions committed to operational database. Application context synchronized with operational database. Committed transactions cleared from application context. Leaves operational database consistent, with conflicting transactions available for manual/interactive resolution.

UC

- Nominal scenario
  - Payment via credit card
  - All correct (credit card number, amount, exp date...)
- Exception scenarios
  - Credit card number incorrect
  - Exp date incorrect
  - Amount too high
  - ...

System design

- Subsystems (software and not software) that compose the system



Styles in RD

- Operational, descriptive
- Formal, semiformal, informal
- System and software

V&V of requirements

- Natural language, UML
  - Inspection, reading
    - By user, by developer
- UML
  - Some syntactic check by tools
- Prototyping
- Formal language
  - Model checking

  - (see V&V chapter)

Tools

- RequisitePro, Doors, Serena RM
- Word, Excel
- UML tools
  - Powerpoint, Visio, specialized tools (StarUML)

Summary

- Goal of requirement engineering is describing *what* the system should do in a requirement document
- Many stakeholders are involved in the process
- Techniques to make the document more precise are
  - Context diagram and interfaces
  - Identifying requirements and classifying them (functional, non functional, domain)
  - Scenarios
  - Use cases

Summary

- Requirements engineering is a key phase
  - Most defects come from this phase, and they are the most disruptive and most costly to fix
- Verification and validation is essential
  - Inspection
  - prototype

Appendix

Domain

- Collection of related functionality

  or

- collection of applications with similar functionality)
  - Ex. banking, that includes subdomains account management, portfolio managemenmt, etc
  - Ex. telecommunication, that includes subdomains switching, protocols, telephony, switching

Application

- Or system
- Software system supporting a specific set of functions. Belongs to one or more domains

Other techniques for RE

Structured presentation

2.6.1 Grid facilities

The editor shall provide a grid facility where a matrix of horizontal and vertical lines provide a background to the editor window. This grid shall be a passive grid where the alignment of entities is the user's responsibility.

Rationale: A grid helps the user to create a tidy diagram with well-spaced entities. Although an active grid, where entities 'snap-to' grid lines can be useful, the positioning is imprecise. The user is the best person to decide where entities should be positioned.

Specification: ECLIPSE/WS/Tools/DE/FS Section 5.6
Source: Ray Wilson, Glasgow Office

Form-based specifications

- Definition of the function or entity.
- Description of inputs and where they come from.
- Description of outputs and where they go to.
- Indication of other entities required.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

Form-based

Insulin Pump/Control Software/SRS/3.3.2	
Function	Compute insulin dose: Safe sugar level
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1)
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose Š the dose in insulin to be delivered
Destination	Main control loop
Action:	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requires	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin..
Post-condition	r0 is replaced by r1 then r1 is replaced by r2
Side-effects	None

Tabular specification

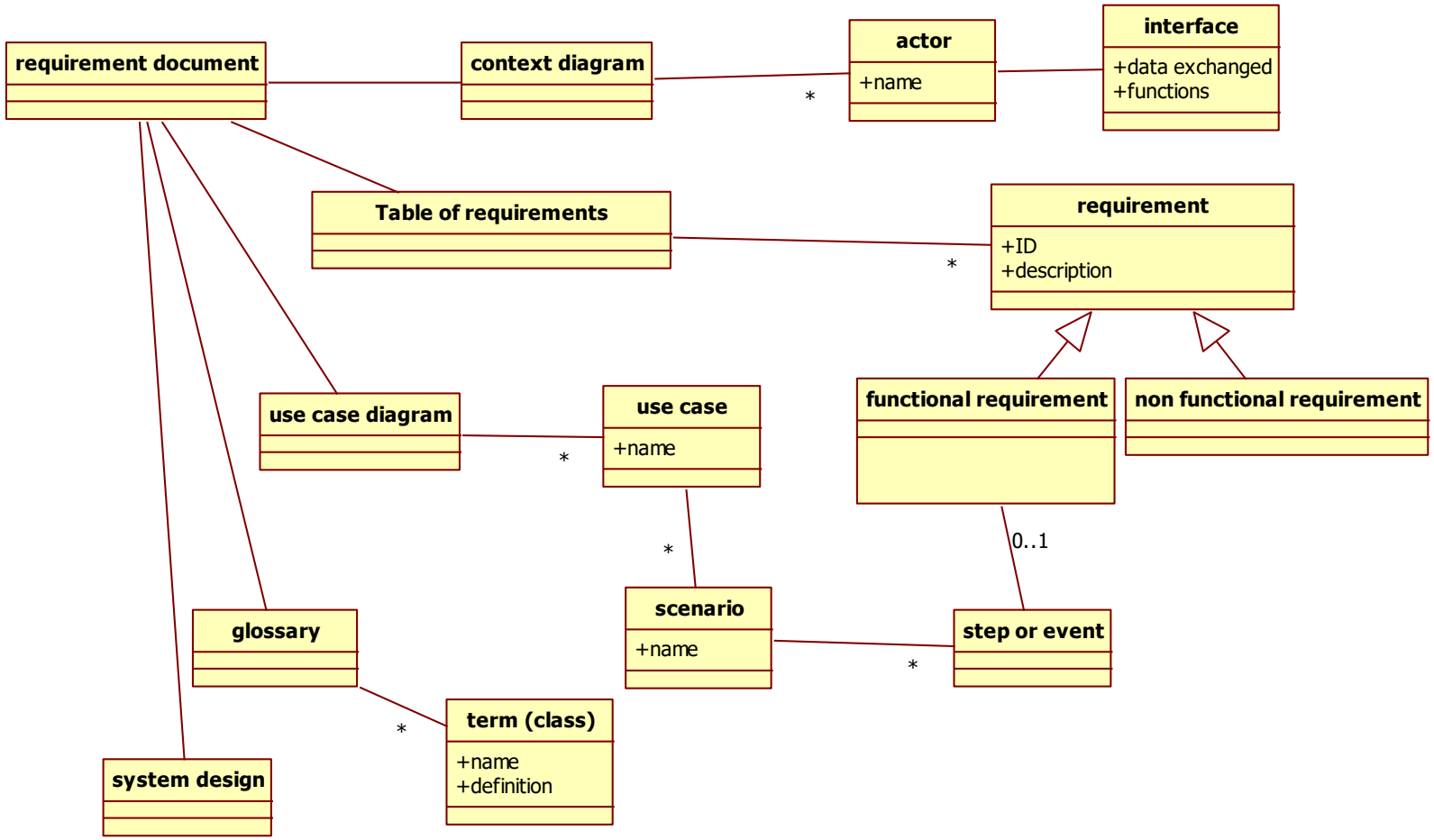
- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.

Tabular specification

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2-r1)<(r1-r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing. ((r2-r1) (r1-r0))	CompDose = round ((r2-r1)/4) If rounded result = 0 then CompDose = MinimumDose

Appendix

- The POS system – requirement document


requirement document
context diagram
actor
+name
interface
+data exchanged
+functions
Table of requirements
requirement
+ID
+description
use case diagram
use case
+name
functional requirement
non functional requirement
glossary
scenario
+name
step or event
term (class)
+name
+definition
system design
*
0..1

Requirements document

1 Overall description
2 Stakeholders
3 Context diagram and interfaces
4 Requirements
- Functional
- Non functional
- Domain

5 Use case diagram
6 Scenarios
7 Glossary
8 System design

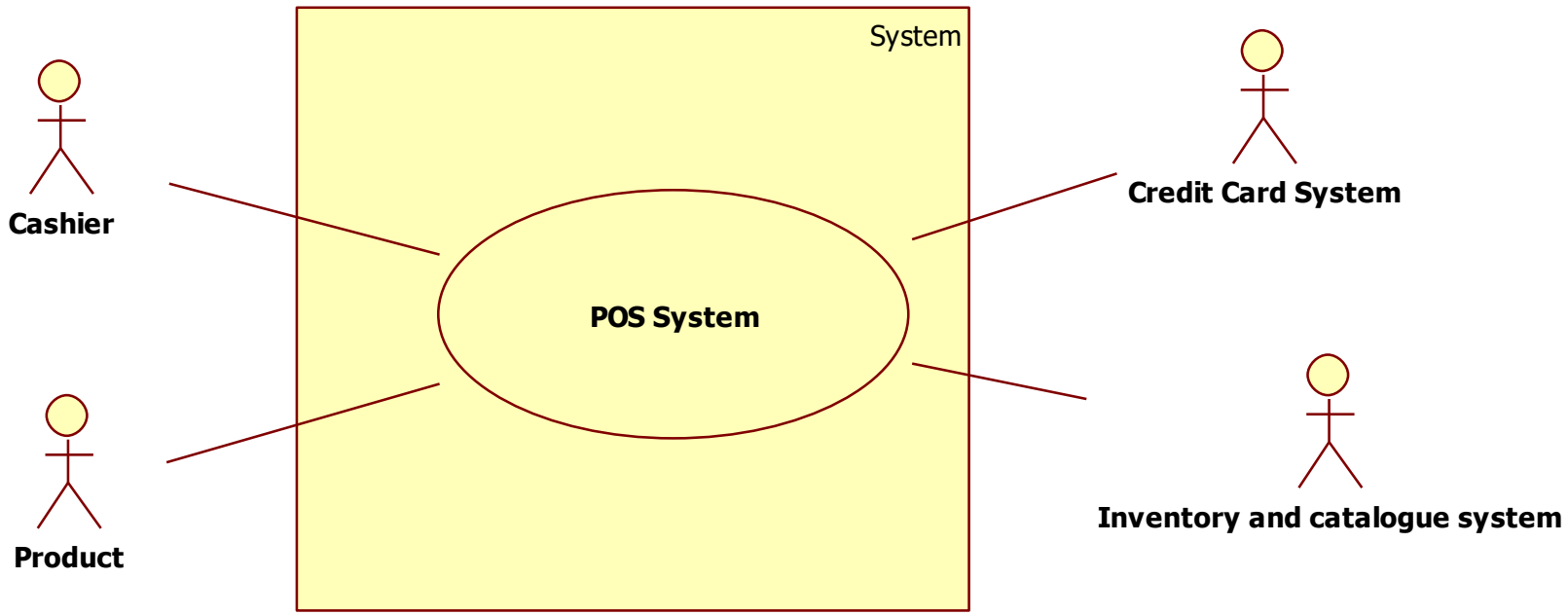
1 Overall description

- A POS (Point-Of-Sale) system is a computer system typically used to manage the sales in retail stores. It includes hardware components such as a computer, a bar code scanner, a printer and also software to manage the operation of the store.
- The most basic function of a POS system is to handle sales
- …

2 Stakeholders

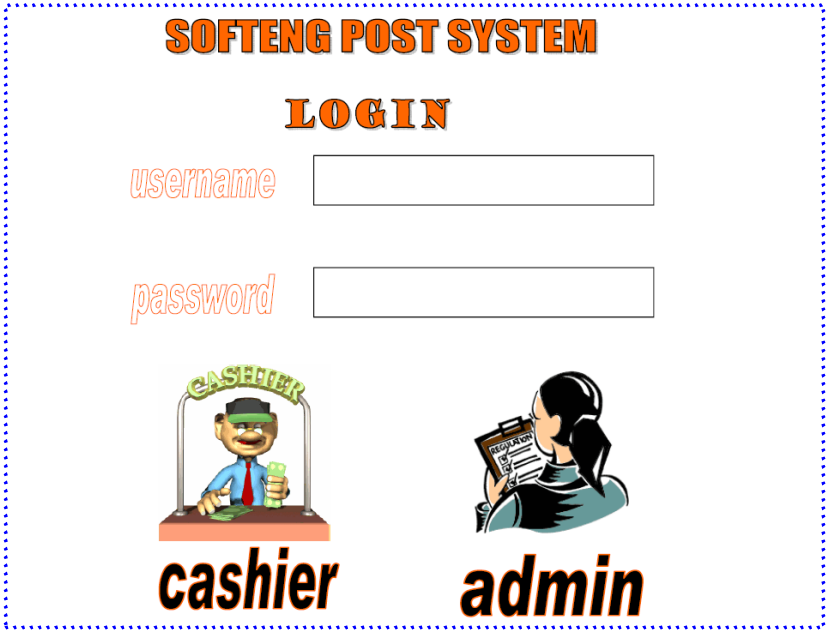
- User
  - Cashier at POS (profile 1)
  - Supervisor, inspector (profile 2)
  - Customer at POS (indirectly through cashier)
- Administrator
  - POS application administrator (profile 3)
  - IT administrator (profile 4)
    - Manages all applications in the supermarket
  - Security manager (profile 5)
    - Responsible for security issues
  - DB administrator (profile 6)
    - Manages DBMSs on which applications are based
- Buyer
  - CEO and/or CTO of supermarket

3.1 Context diagram



3.2 GUI interface

- Sketch of interface, typically built with GUI builder



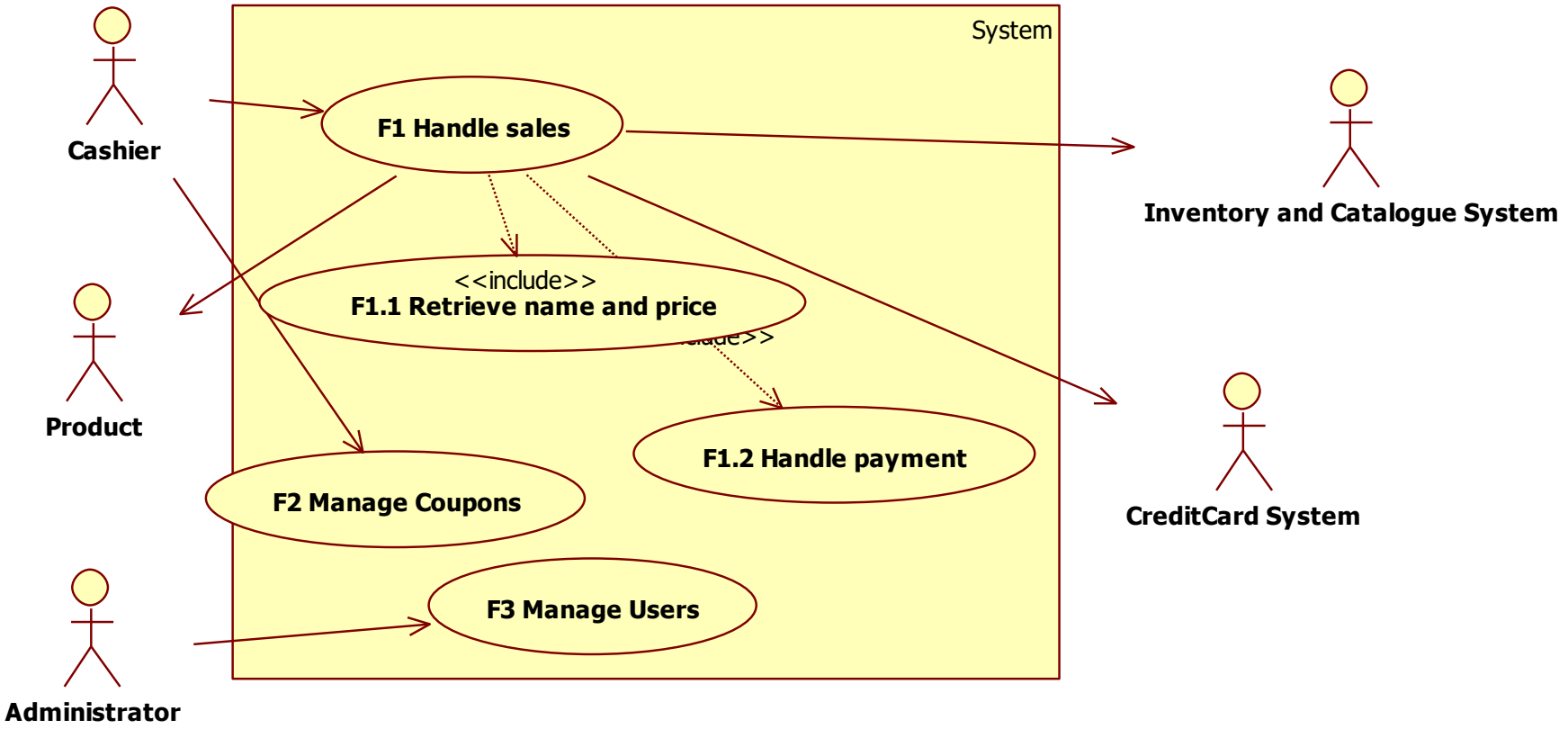
4.1 Functional requirements

Rquirement ID	**Description**
F1	Start sale transaction
F2	End sale transaction
F3	Log in
F4	Log out
F5	Read bar code

4.2 Non functional requirements

Requirement ID	Description
NF1(efficiency)	Function F1 less than 1msec
NF2 (efficiency)	Each function less than ½ sec
Domain1	Currency is Euro – VAT is computed as ..

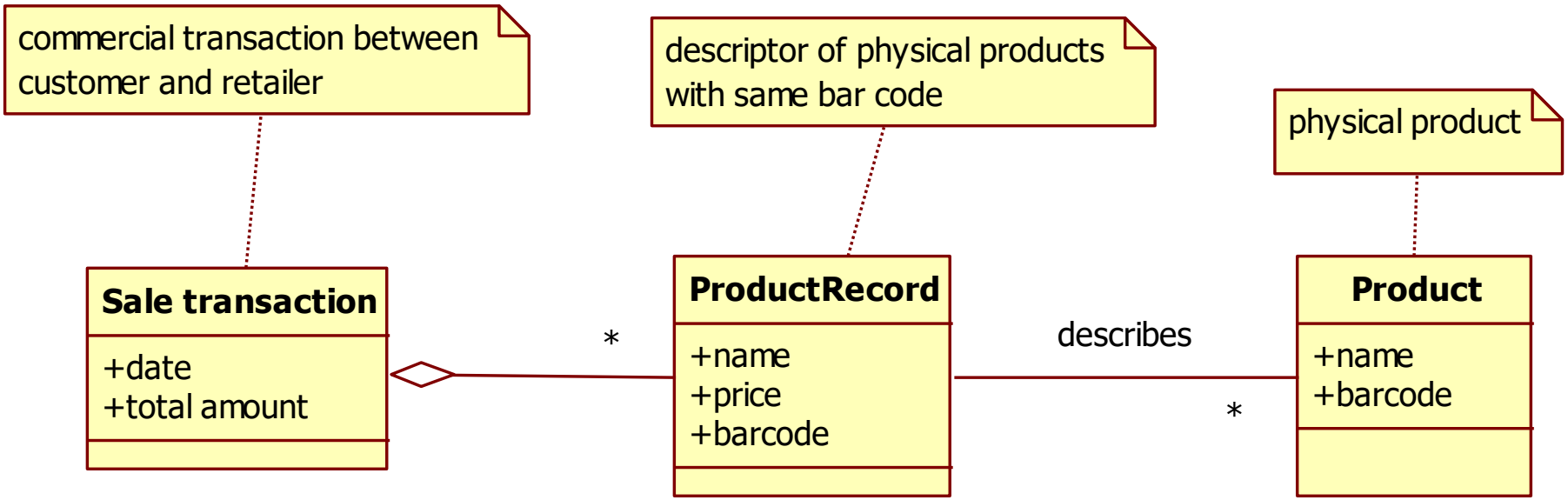
5 UCD


System
Cashier
F1 Handle sales
Inventory and Catalogue System
<<include>>
F1.1 Retrieve name and price
Product
F1.2 Handle payment
F2 Manage Coupons
CreditCard System
F3 Manage Users
Administrator

6 Scenarios

Step	Description
1	Start sales transaction
2	Read bar code
3	Retrieve name and price given barcode
	Repeat 2 and 3 for all products
4	Compute total
5	Manage payment cash
6	Deduce stock amount of product
7	Print receipt
8	End sales transaction

7 Glossary


commercial transaction between customer and retailer
descriptor of physical products with same bar code
physical product
Sale transaction
+date
+total amount
*
ProductRecord
+name
+price
+barcode
describes
*
Product
+name
+barcode

8 System design