# Requirement Engineering
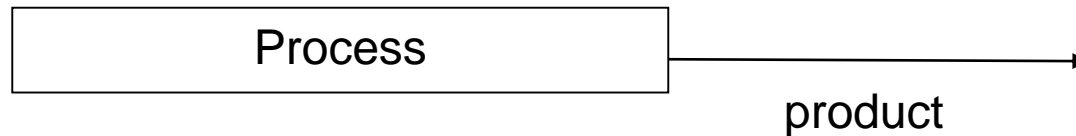
# RE

- Requirement engineering is about defining the <u>product properties</u> before starting development

| Process |
|---|

product →

# Without RE

- Product properties are unclear
- Testing is not feasible

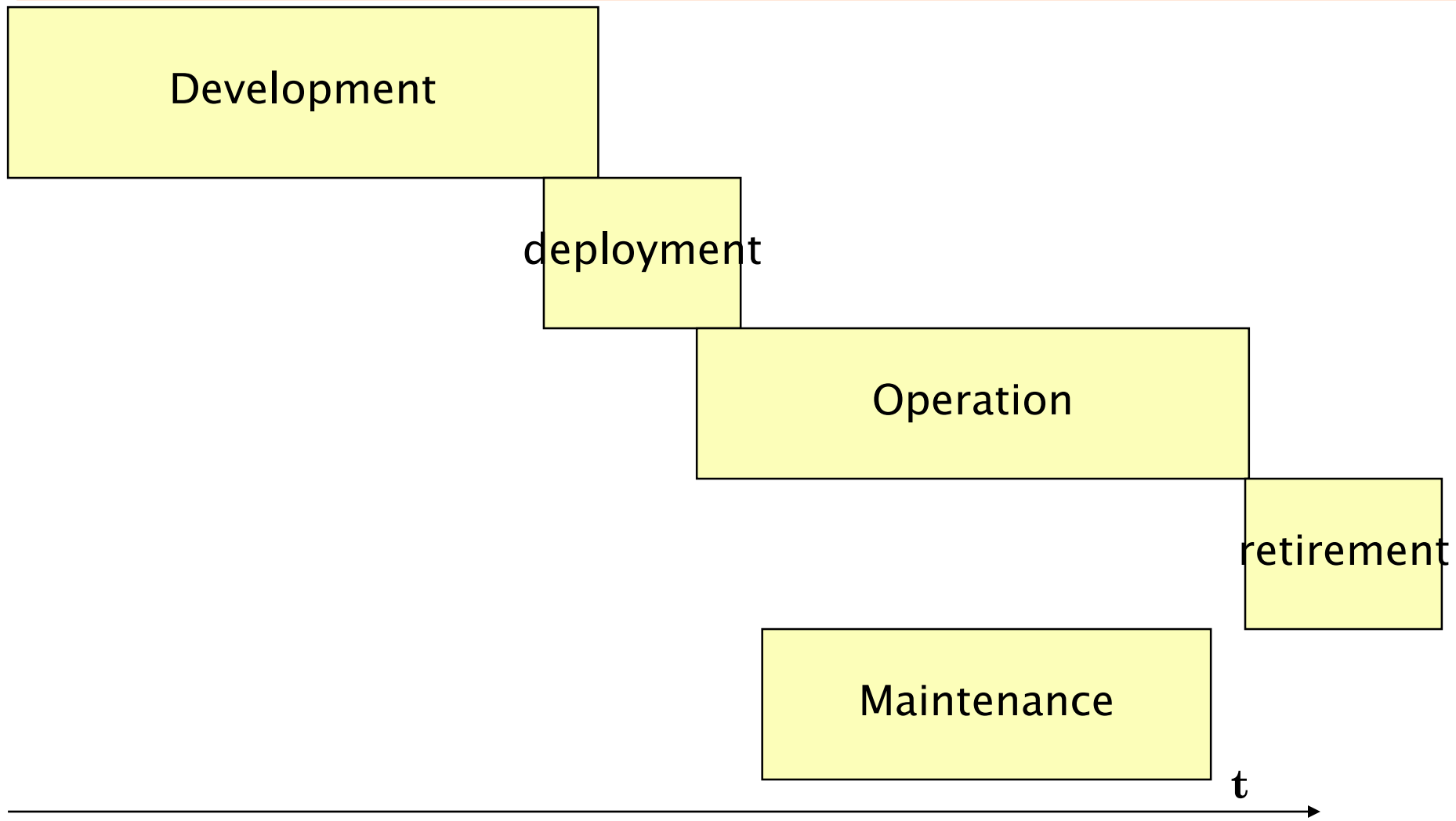# Product properties

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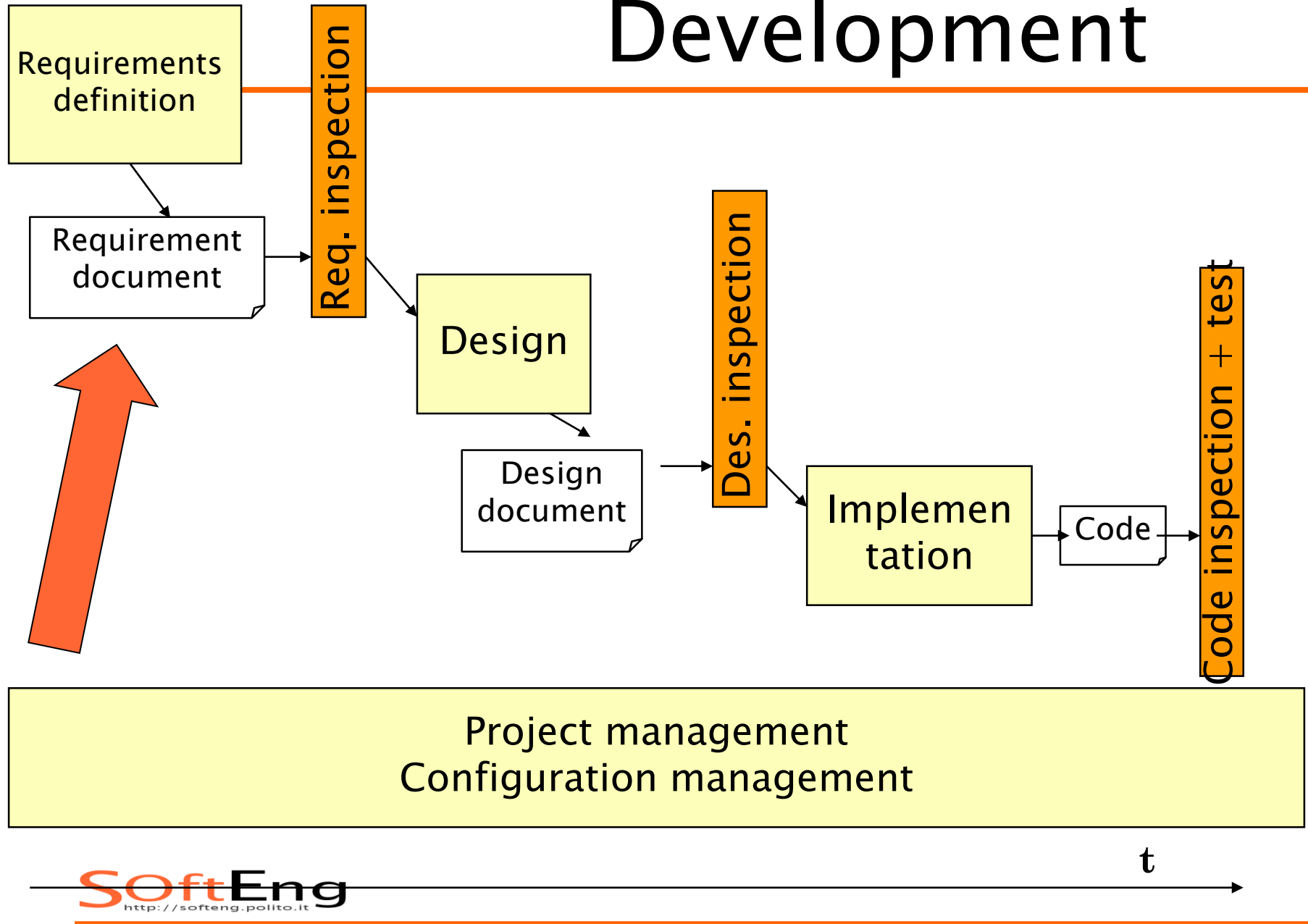    - Maintainability
    - Security
    - ...

# Outline

- Concepts and definitions
- Techniques for formalization
- Techniques for elicitation
- Techniques for V and V

# Main Phases

Development

deployment

Operation

retirement

Maintenance

t

# Development

Requirements definition

Requirement document

Req. inspection

Design

Design document

Des. inspection

Implementation

Code

Code inspection + test

Project management
Configuration management

t

SoftEng
http://softeng.polito.it

# Requirement engineering

elicitation

Informal description

Analysis and formalization

Requirement document

Inspection, reading

analyst

stakeholders
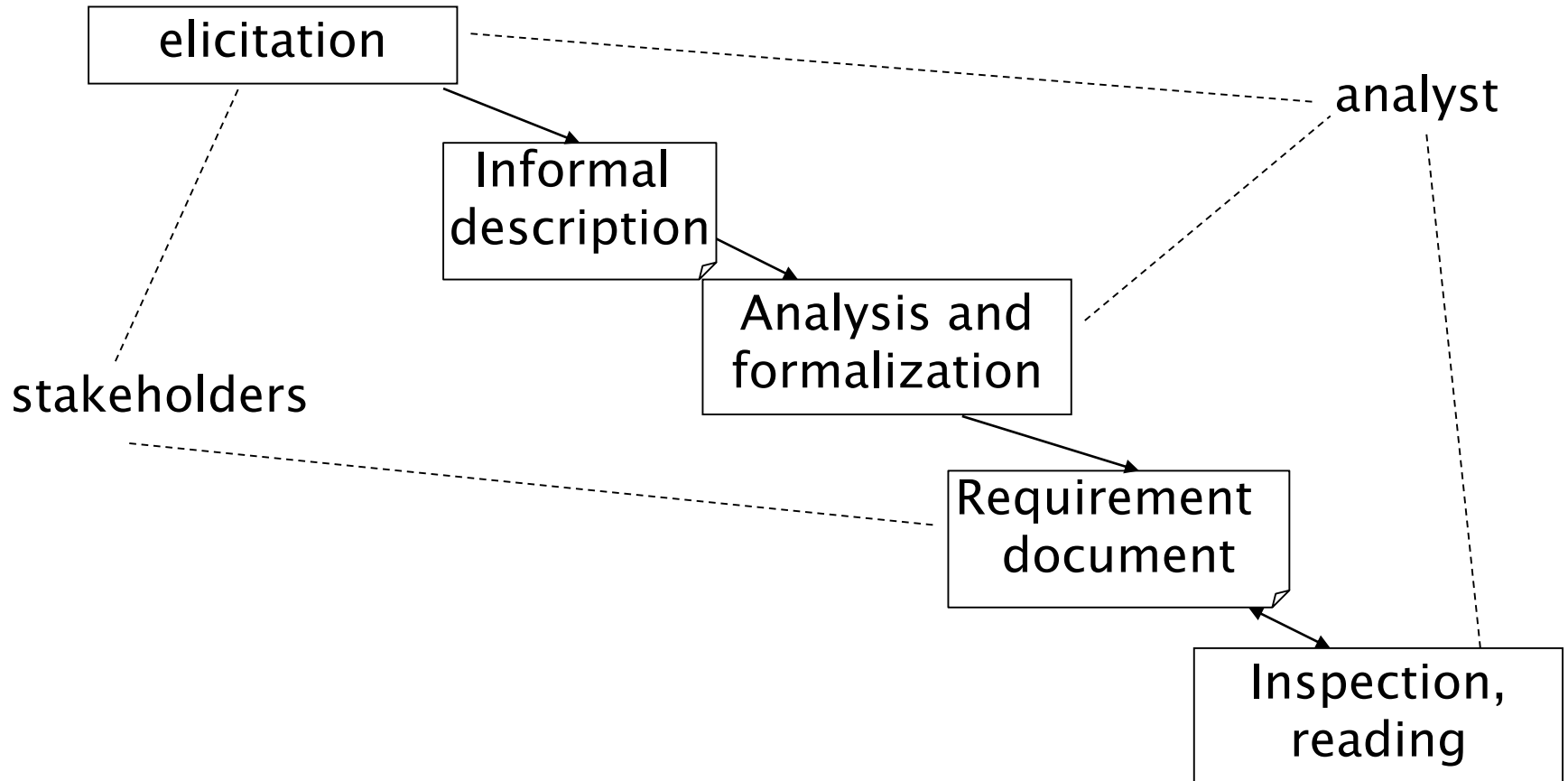
# Requirement

- Description of product property

  ◆ Functional – not functional

- The final product may, or may not, have properties that match the requirements

Requirements

Properties

# Requirements vs. properties



Requirements → Software product

properties →

Requirements are requests, that may or may not become properties of the product

- Ex.

- Requirement: response time of all functions $<0,5$ sec

- Property: response time of functions between 2secs and 10 secs

# Starting point

- Usually an informal description by a client or a potential user to the developer
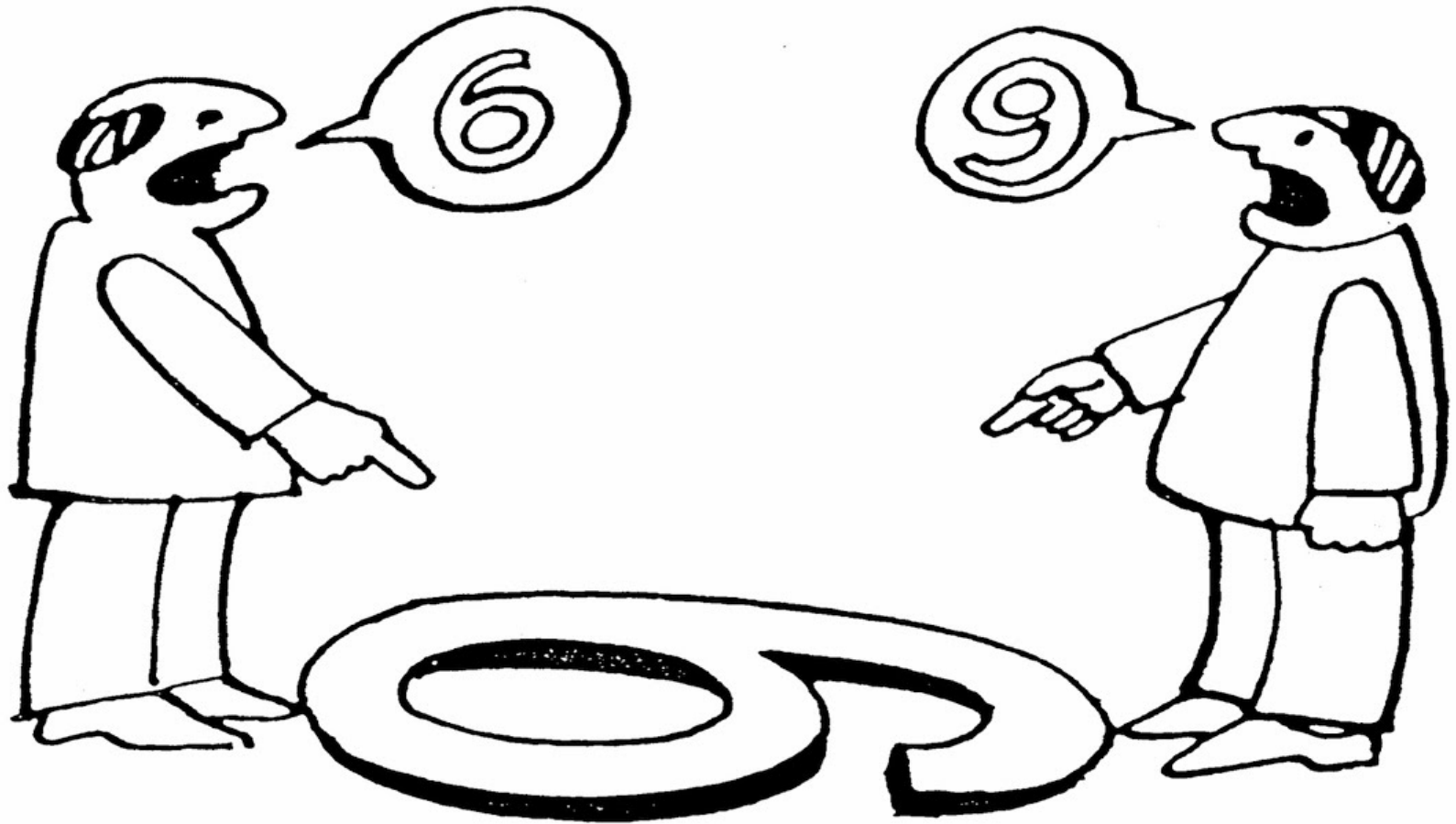
Client                                              Developer

# 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? Exchange of item? Payment?

# Completeness and consistency

- In principle, requirements should be both complete and consistent.
  - ◆ Complete
    - They should include descriptions of all features required.
  - ◆ Consistent
    - There should be no conflicts or contradictions in the descriptions of the system features.
  - ◆ 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

# Techniques – formalization

- Stakeholders
- Personas, stories
- Context diagram and interfaces
- Requirements, F, NF
- Scenarios, sequence diagrams
- Use cases
- Glossary

# Stakeholder

- Basic idea: ask the client an (informal) description of the product to build

- Who exactly is the 'client'?

-

# Stakeholder

- Role person company or system that is directly or indirectly involved in the project and who may affect or get affected by the outcome of the project

- User
  - Uses the system
  - Can include different user profiles

- Buyer / commissioner
  - 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

# Stories and personas

- A technique to informally define what the system should do

- Best suited for mass market applications where 'end user' has little meaning

- AKA profiling / segmentation in marketing, adapted to software products

# Profiling / segmentation

- Characterize people under a number of dimensions, to better define and sell targeted products / services

**Demographic profile**

- Age
- Gender
- Ethnicity
- Income
- Level of education
- Religion
- Profession/role in a company

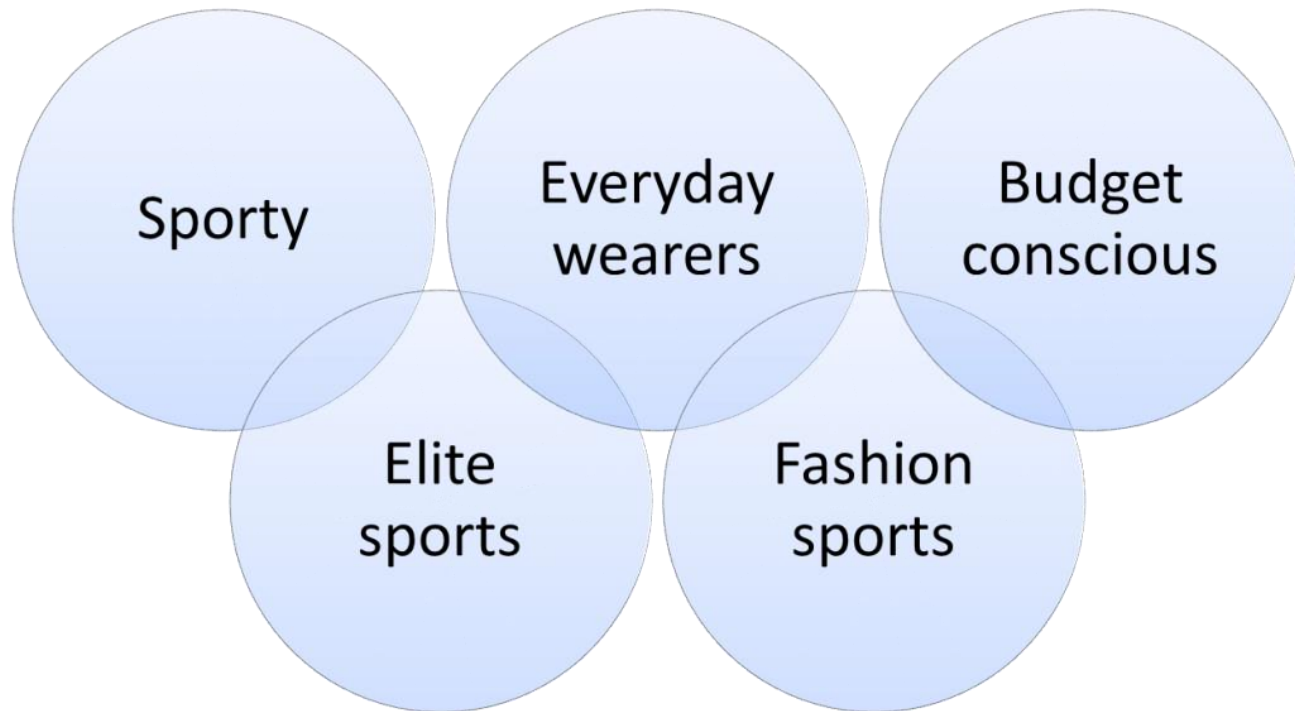**Psychographic profile**

- Personality traits
- Hobbies
- Life goals
- Values
- Beliefs
- Lifestyles

**Geographic profile**

- Country
- Region
- City
- Postal code

# Ex market segments

In garments business, segments could be:



Sporty

Everyday wearers

Budget conscious

Elite sports

Fashion sports

# Profiling in e-commerce

Web site behaviour

- Number of sessions website
- Number of pages visited
- Time spent on site
- URLs visited
- Page types visited
- Shopping cart value
- Referral source
- Inactivity

Web site behaviour – 2

- First time visitor
- Returning visitor
- Returning customer

# Use of profiles

- **First time visitor**
  - ◆ Offer x% discount on first order
- **Returning visitor**
  - ◆ Remind of x% discount
- **Returning customer**
  - ◆ Propose products similar to ones previously bought

# Personas

- Identify and describe typical users
- A persona is a subset of actor (see later)

  - Ex persona1: female, middle age, professional, high income, married with children
  - Ex persona2: male, young, student, low income, not married

# Personas

- For each persona describe life scenarios
  - Persona1, work day: wake up, breakfast, drive children to school, drive to office ...
  - Persona1, week end day: ...

- For each persona /scenario identify possible interaction with application/object

# Example

## User Persona Type

| Trait 1 | Trait 2 | Trait 3 | Trait 4 |

*"A quotation that captures this user's personality."*

**Age:** 1-100
**Work:** Job title
**Family:** Married, kids, etc.
**Location:** City, state
**Character:** Type

### Personality

| Introvert | | Extrovert |
| Thinking | | Feeling |
| Sensing | | Intuition |
| Judging | | Perceiving |

### Goals

- A task that needs to be completed.
- A life goal to be reached.
- Or an experience to be felt.

### Frustrations

- The challenges this user would like to avoid.
- An obstacle that prevents this user from achieving their goals.
- Problems with the available solutions.

### Bio

The bio should be a short paragraph to describe the user journey. It should include some of their history leading up to a current use case. It may be helpful to incorporate information listed across the template and add pertinent details that may have been left out. Highlight factors of the user's personal and professional life that make this user an ideal customer of your product.

*Remember - you may modify this template, remove any of the modules or add new ones for your own purpose.*

### Motivation

Incentive

Fear

Growth

Power

Social

### Brands & Influencers

FAKE BRAND    FAKE BRAND    FAKE BRAND

### Preferred Channels

Traditional Ads

Online & Social Media

Referral

Guerrilla Efforts & PR

# Stories

- Help to understand interactions

- Are a cheap way to illustrate design solution from user's (persona's) point of view

- Tell user's goals, motivations and actions

# Stories

- "What should this product do?"

- "How would user behave in this context?"

- "What if...?"

# Stories

- Without your solution present-based
  - Focus is set on current practices that illustrate 'state of the art' and the problem context

- With your solution future-based
  - Focus on how problems could be addressed (without diving into details and jargon)

# Stories

- In what settings will the product be used?

- Is the persona frequently interrupted?

- With what other products will it be used?

- What primary activities does the persona need to perform to meet her goals?

- What is the expected end result of using the product?

# Example: EZGas

- Persona 1: high income professional, male, married, with children, 45 yo
  - Story: need to visit a client, morning in rush hour, is late, need to find <u>closest</u> open station, no matter what price
- Persona 2: <u>doctor</u>, female, 35
  - Story: back from long hospital shift, 2 am, alone, need to find <u>safest</u> place to refuel

# Example: EZGas – 2

- Persona 3: student, male, 22, low income
  - Story: borrowed car from a friend, need to find <u>cheapest</u> place to refuel
- Persona 4: young professional, female, 30, is very sensitive about pollution and green house emissions, has hybrid car

  - Story: commutes home every weekend, needs to find more <u>sustainable</u> fuel provider
- Persona 5: similar to 4, has full electric car
  - Story: needs to find <u>charging station</u>

# In summary

- Personas + stories help to find

  - Functionalities to be offered to satisfy categories of needs (profiles / personas)
  - (functionalities NOT to be offered)

  - Interaction modalities
    - Ex mobile usage on car vs desktop usage at home

# Exercise

- Define Personas /stories for RCV

# Context diagram

- To understand 'in' and 'out'

# 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 (can specialize in one or more personas)

- Defines **interfaces** between inside and outside

# Context diagram
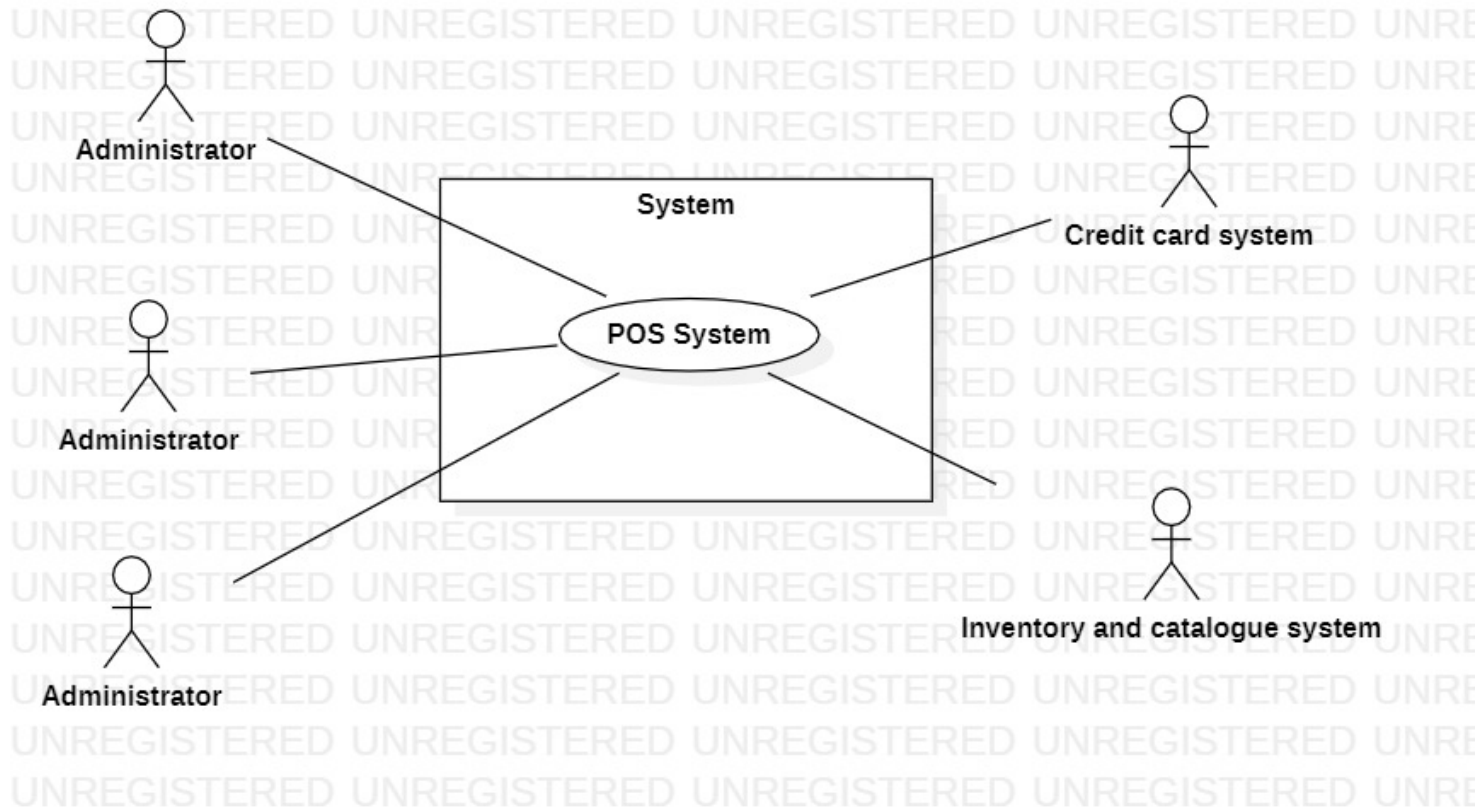
# Interfaces

- **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, ex http/REST + json)

# Interfaces – tabular form

| Actor | Physical interface | Logical interface |
|---|---|---|
| Cashier | Screen, keyboard | Graphical User Interface (to be described, ex slide 46) |
| Product | Laser beam | ReadBarCode (to be specified, ex slide 44) |
| Credit card system | Internet connection | API description (ex https://developer.visa.com/docs for ViSA APIs |
| Administrator | Screen, keyboard | Graphical User Interface + command line interface |

# 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.

SOftEng
http://softeng.polito.it

# 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
```

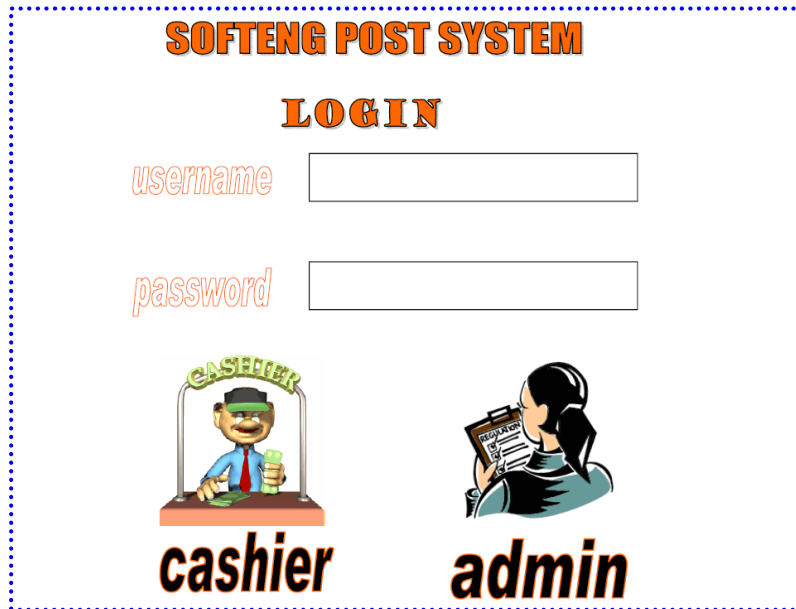# Data interface (XML)

```
<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 requirements

- Techniques
  - Clearly separating and numbering requirement

# 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 |

# Functional requirements

- **Techniques**
  - Clearly separating and numbering requirements
  - Hierarchical numbering

# Functional

| Requirement ID | Description |
|---|---|
| F1 | Handle sale transaction |
| F1.2 | Start sale transaction |
| F1.3 | End sale transaction |
| F2 | Authorize and authenticate |
| F2.1 | Log in |
| F2.2 | Log out |
| F2.3 | Define account |

# Naming the FR is useful for

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

# ISO 9126 / ISO 25010

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

- ◆ Usability
  - – Effort needed to learn using the product (installation, day to day usage)
  - – Satisfaction expressed by the user
  - – Existence of functions needed by the user
- ◆ Efficiency
  - – For a given function in a given context: response time
  - – For a given function / for a complete product: memory / cpu / bandwith / energy used

- ◆ Correctness
  - – Capability to provide intended functionality in ALL cases
- ◆ Reliability / availability
  - – Defects visible by end user per time period / Probability of defect over a time period
  - – Percentage of time the product is / is not available to end user

- ◆ **Maintainability**
  - – Effort (person hours) needed to add /modify / cancel a software function
  - – Effort to fix a defect
  - – Effort to deploy on a different platform (DB, OS, ..)
- ◆ **Portability**
  - – Effort to redeploy application on another platform
    - – Os, database, network, screen

- ◆ Security
  - – Protection from malicious access
  - – Access only to authorized users
  - – Sharing of data
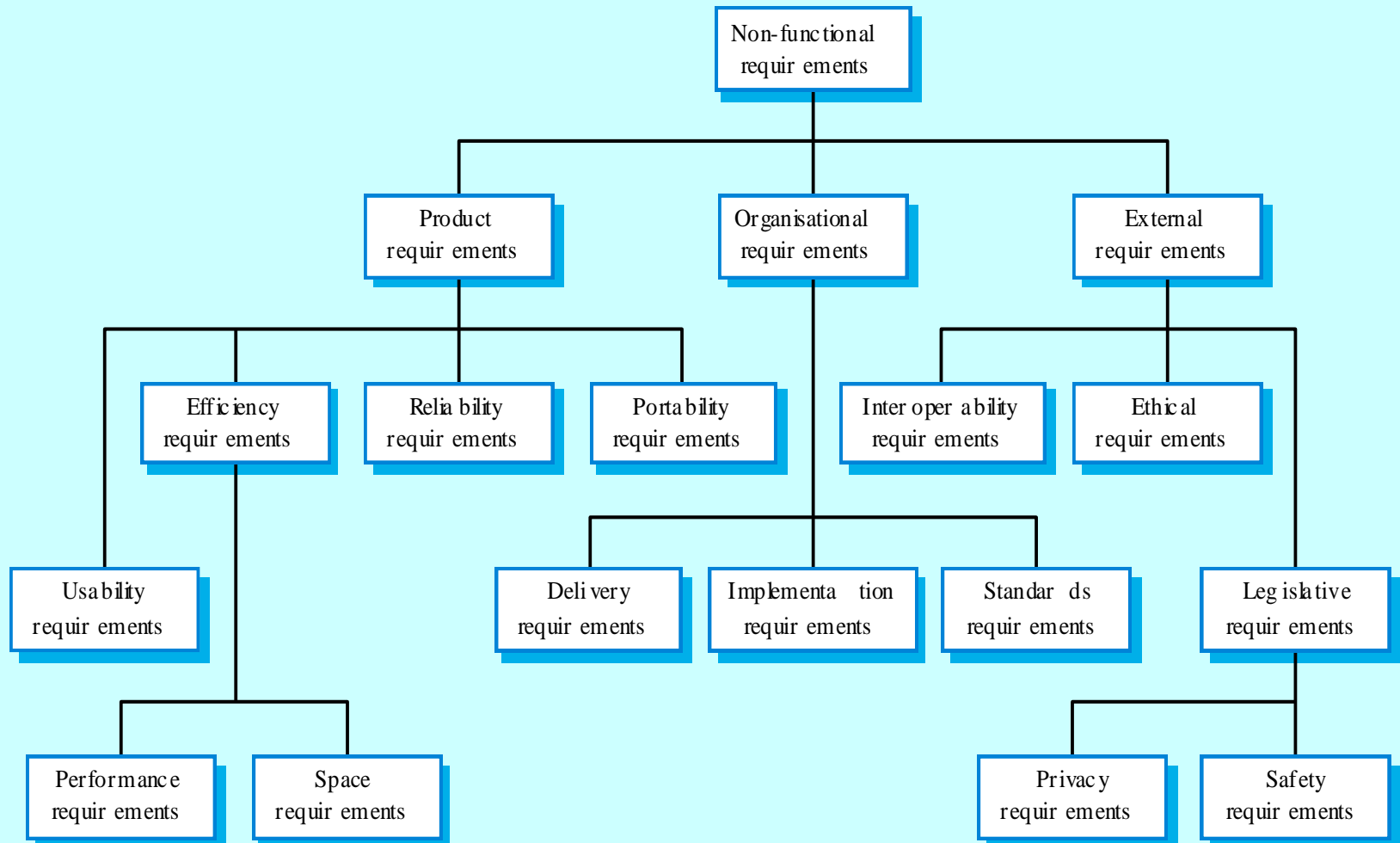
  GDPR (EU)
  CCPA (California)

- ◆ Safety
  - – Absence of harm to persons
  - – Absence of hazardous situations for persons

- ◆ Dependability
  - – Safety + security + reliability

# 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 <u>should be easy to use</u> by experienced controllers and should be organised in such a way that user <u>errors are minimised</u>.

- **Measurable**
  - Experienced controllers shall be able to use all the system functions after a total of <u>two hours training</u>. After this training, the average number of errors made by experienced users shall not exceed <u>two per day</u>.

# Measures for NF reqs

| Property | Measure |
|---|---|
| Speed | Processed transactions/second<br>User/Event response time<br>Screen refresh time |
| Size | M Bytes<br>Number of ROM chips |
| Ease of use | Training time<br>Number of help frames |
| Reliability | Mean time to failure<br>Probability of unavailability<br>Rate of failure occurrence<br>Availability |
| Robustness | Time to restart after failure<br>Percentage of events causing failure<br>Probability of data corruption on failure |
| Portability | Percentage of target dependent statements<br>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 <u>not testable</u>, and therefore <u>useless</u>

# 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?

# Requirements interaction

- **Efficiency vs security**

    - Efficiency → minimize number of software components and calls between them

    - Security → add components (and layers and calls)

# Requirements interaction

- In the end <u>key stakeholders</u> must decide the ranking of NF requirements
  - ◆ More important security or efficiency?

- This decision is a <u>business</u> decision (not a technical one) and should NOT be taken by developers

# 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 $9.81ms^2$ * compensated gradient/alpha and where the values of $9.81ms^2$ /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.
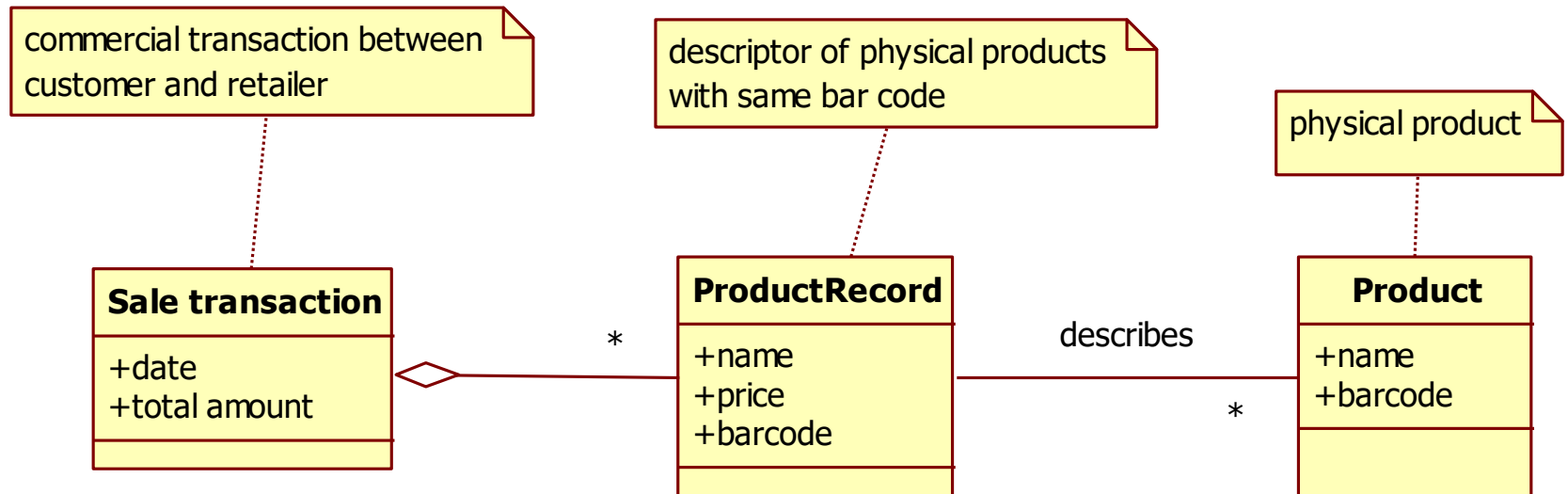
# Glossary

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

- Product = ..

# Class diagram

- See UML slides
- Can be used
  - To refine glossary

  - To describe application model
  - To describe system design

# Glossary with Class diagram

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

**Product**

+name
+barcode

\*

describes

\*

# 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

# Scenarios and Use cases

# Scenario

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

- Sequence: <u>time is key part</u> of this model

  - Time is NOT part of Functional requirements, Glossary, Context diagram, Use cases

# Scenario sale1

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

# Sale1

- Precondition
  - Cashier is authenticated

# Scenario login

| Step | Description |
|------|-------------|
| 1 | Cashier starts application |
| 2 | Application asks account name, pwd |
| 3 | Cashiers enters account name, pwd |
| | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |

# Scenario login

- Precondition: cashier is not authorized
- Post condition: cashier is authorized and authenticated

# Other scenarios

- (payment cash)
- Payment with credit card
- Payment with loyalty card

- Payment does not succeed, abort sale
- Customer refuses an item

# Scenario / pre post conditions

- **Precondition**
  - Condition to be satisfied before starting the scenario
    - Ex: bar code X is valid
    - Ex: bar code X corresponds to a product available in the shop

- **Postcondition**
  - Condition satisfied at end of the scenario
    - Ex: amount in cash after transaction = amount in cash before transaction + T

# Use case

- Set of scenarios with common user goal

- Ex: use case:  Handle sales
  - Scenario1: sell 2 products
  - Scenario2: sell 3 products
  - Scenario3: sell n products, abort sale because customer has no money
  - Scenario 4: sell n products, customer changes one of the products

- Ex other use cases
  - Handle coupons
  - Handle users

# Use case diagram



System

Cashier

Product

Administrator

Inventory and Catalogue System

CreditCard System

F1 Handle sales

<<include>>
F1.1 Retrieve name and price

F1.2 Handle payment

F2 Manage Coupons

F3 Manage Users
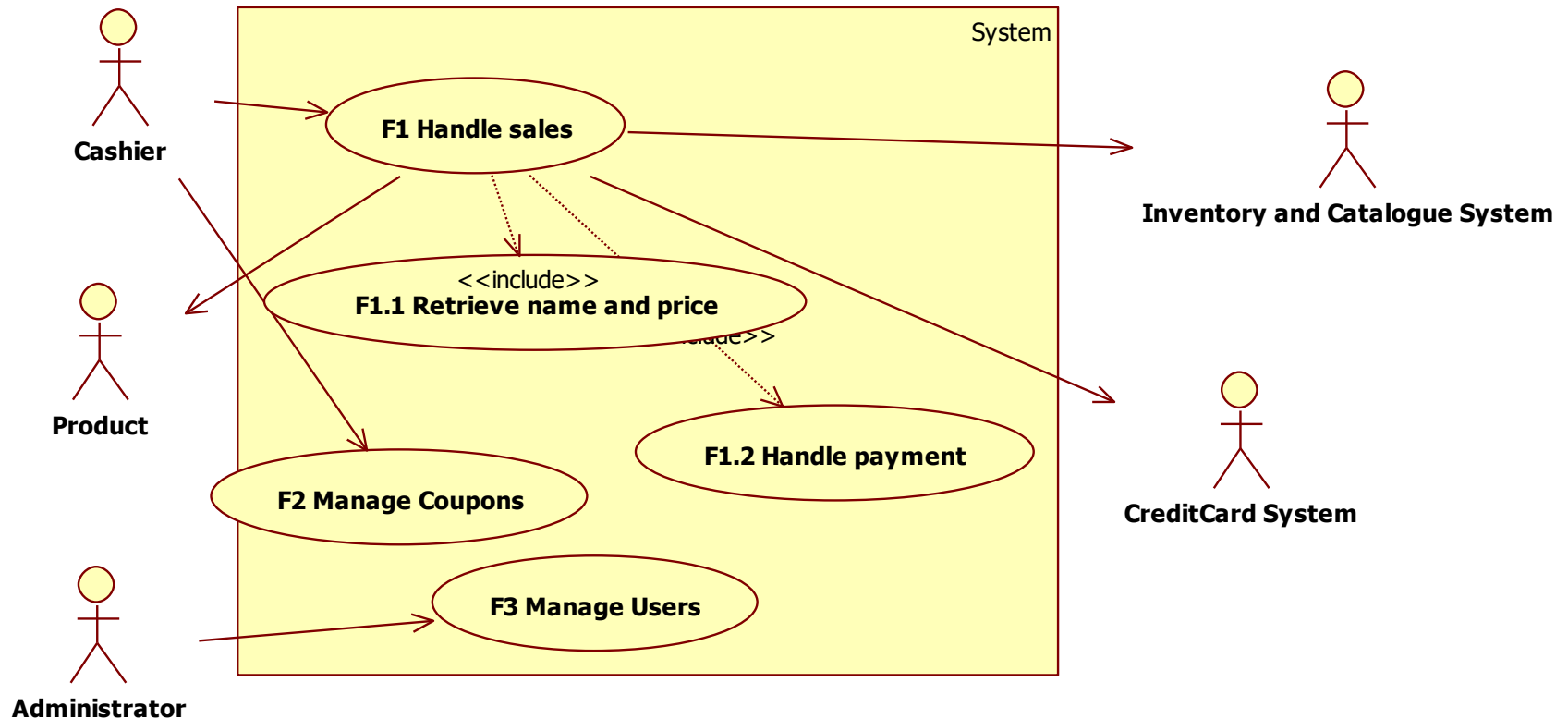
# Use case

- 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.**

  - ◆ Use case similar to Class  (Model)
  - ◆ Scenario similar to instance of class

# Key elements



GOAL

System

User (Actor)
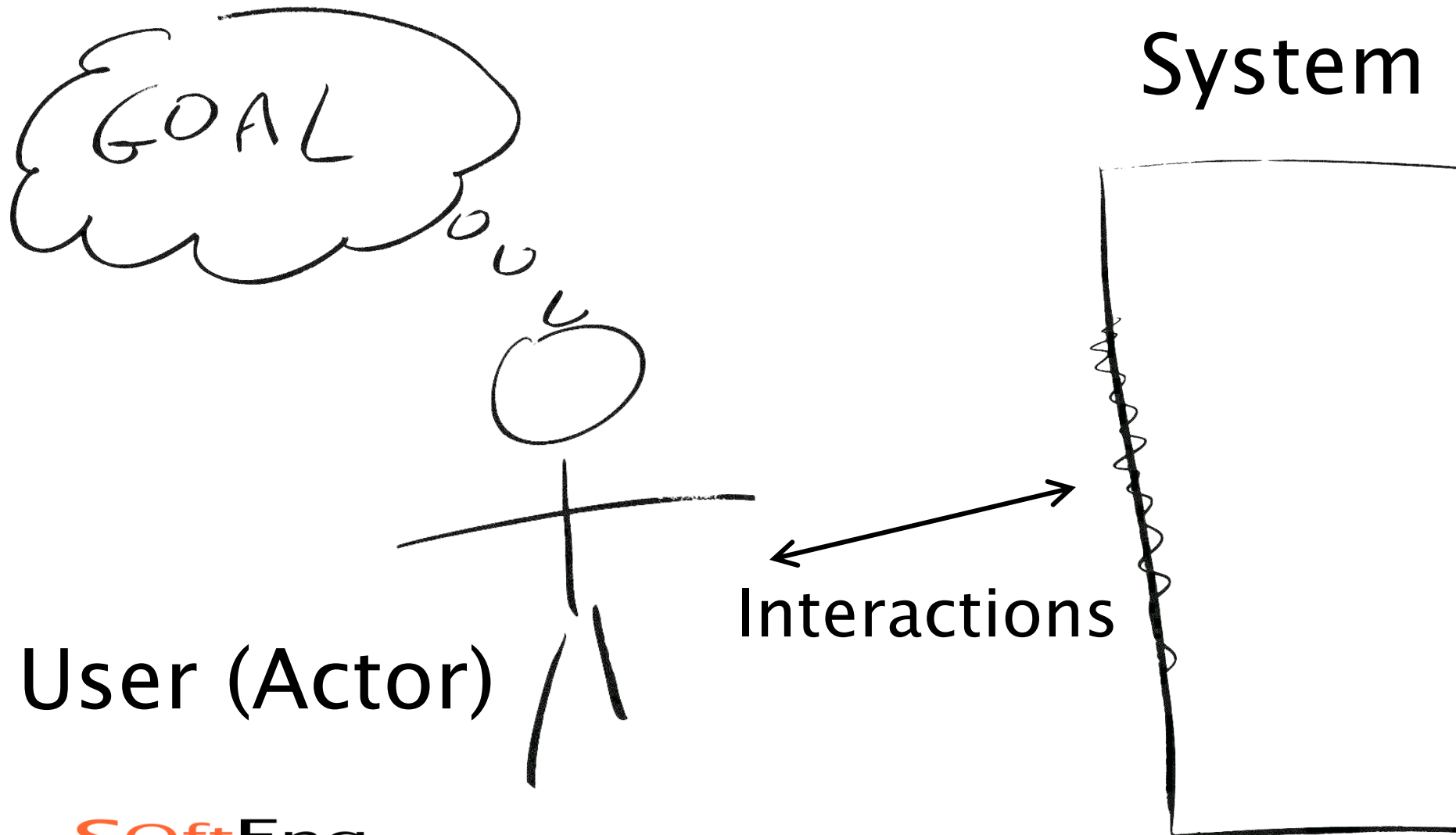
Interactions

# 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

As a *&lt;actor type&gt;*

bank customer

I want *&lt;to do something&gt;*

to perform a withdrawal

So that *&lt;some value is created&gt;*

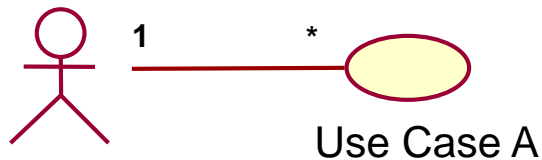I get some cash for me

# 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
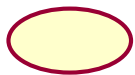
**Use Case**

- A functional unit (functionality) part of the system

# Relationships



Actor — 1 — * — Use Case A

Actor → Use Case A

Actor ← Use Case B
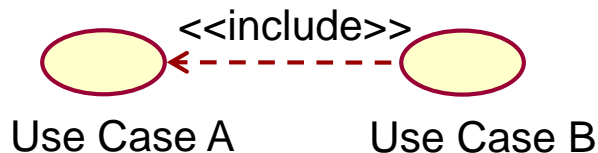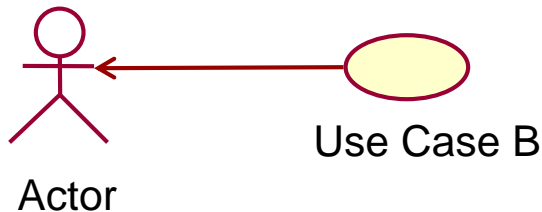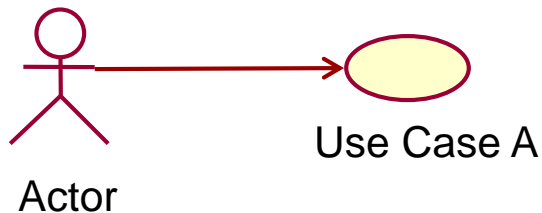
Use Case A ←<<include>>-- Use Case B

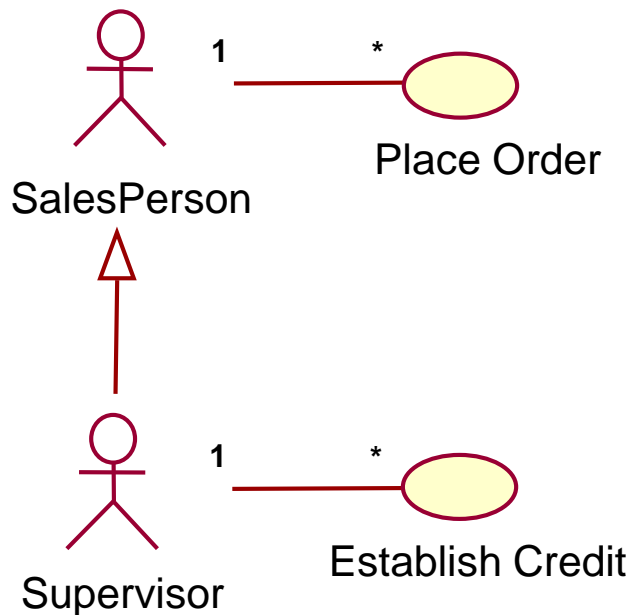- **Association models:**
  - Which actors participate in a use case
  - Where execution starts
  - Adornments (e.g. multiplicity, direction) allowed
  - Actor1 participates in Use CaseA and is the trigger of the use case
  - Actor2 participates in UseCaseB and UseCaseB is the trigger

- **Include**
  - Models that functionality A is used in the context of functionality B (one is a phase of the other)

# Relationships: generalization



**Use Case A**     **Use Case B**



**SalesPerson**     1    *   **Place Order**



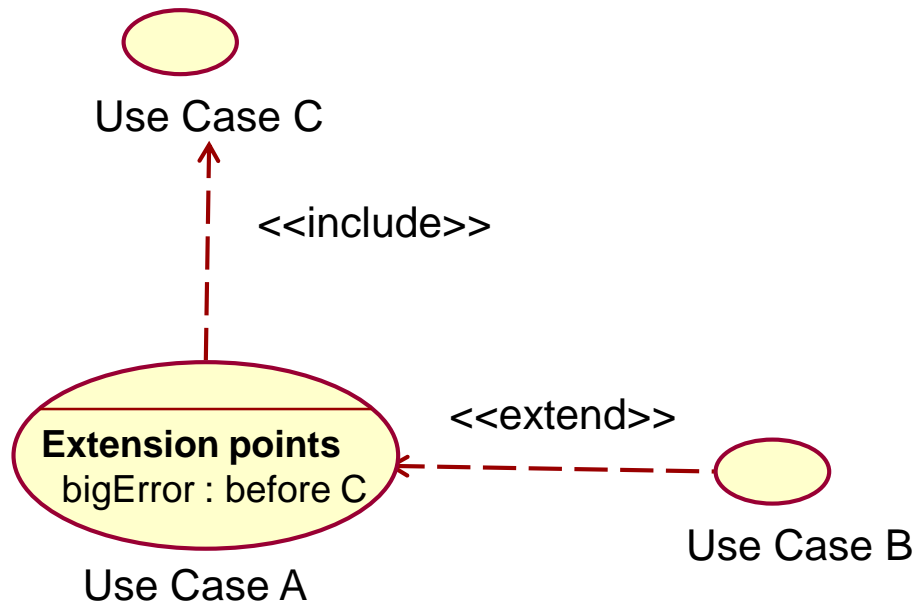**Supervisor**     1    *   **Establish Credit**

- 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
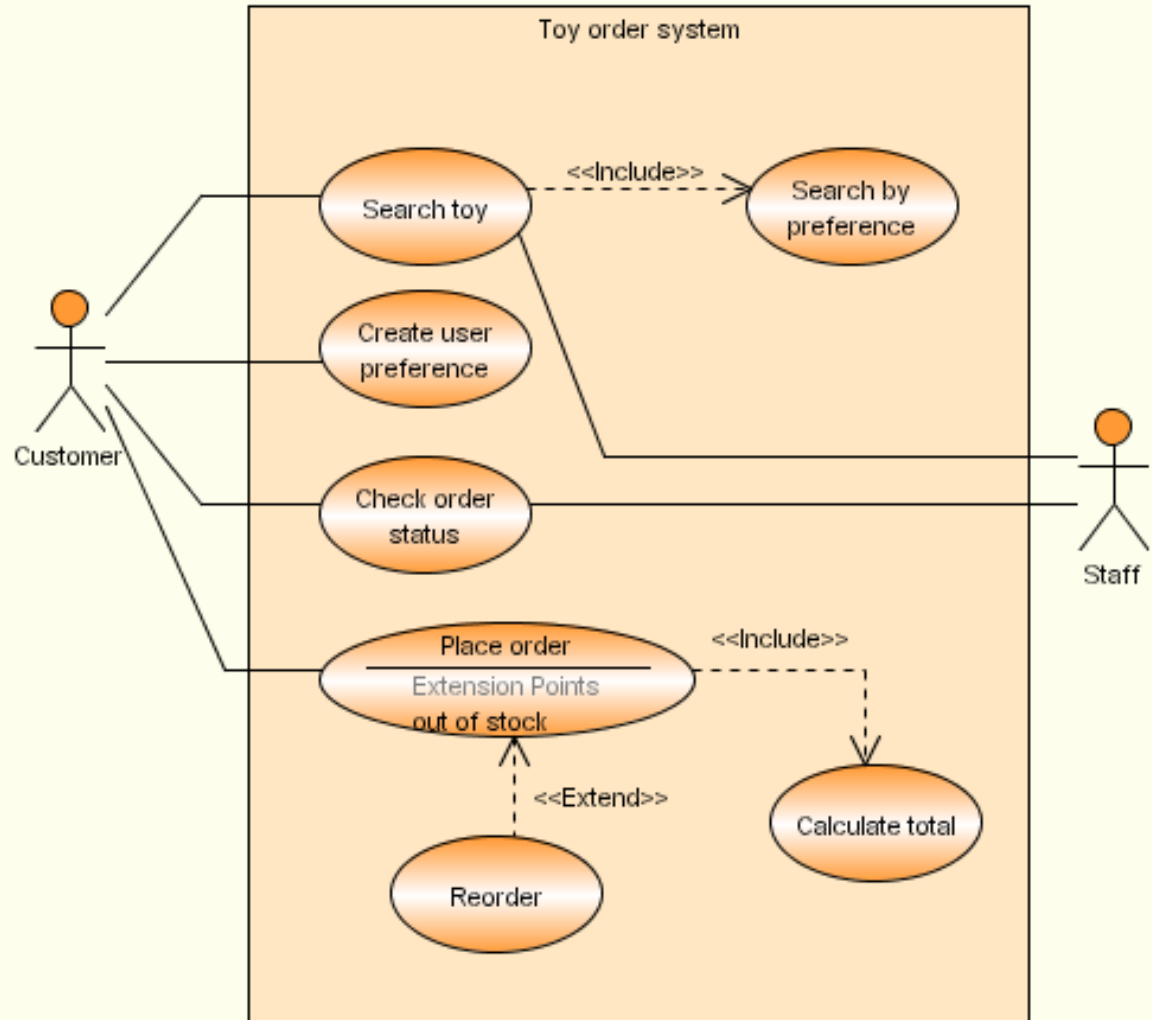
# Relationships: extension



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

# Use case – Example

# 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.

# Use cases vs.requirements

- Requirement (functional)

- Use case
  or
  scenario in use case
  or
  step in scenario

- Mapping is not 1:1

- Requirement purpose is to support traceability and tends to be finer grained than use case

- Use case purpose is to understand how system works

# Use cases vs requirements

- High level requirement (ex F1, F2)

- Low level requirement (ex F1.1  F1.2)

- Use case that includes other use cases

- Use case <included> by other use case

# Example: student management

- students select courses
- professors update the list of available courses
- professors plan exams for each course
- professors can access the list of students enrolled in a course
- professors perform exams then record issue of exam for student (pass/no pass, grade)
- all users should be authenticated

# Example

# Example



Check availability

Book Hotel

<<include>>

<<include>>

Acquire Customer data

Customer

<<include>>

Acquire Credit Card data

Book Hotel with Credit Card

# Use case diag and class diagram

- They must be consistent
  - Use case diagram
    - actor
    - use case



    - interaction
  - Class diagram
    - may become a class
    - must become one operation on a class – may originate several operations on several classes (see sequence diag)
    - not represented (see dynamic diagrams)

# 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
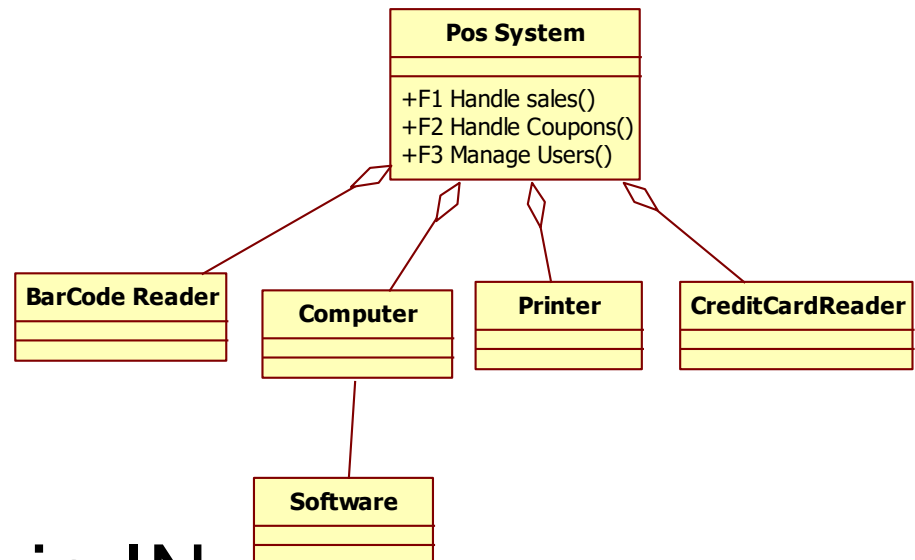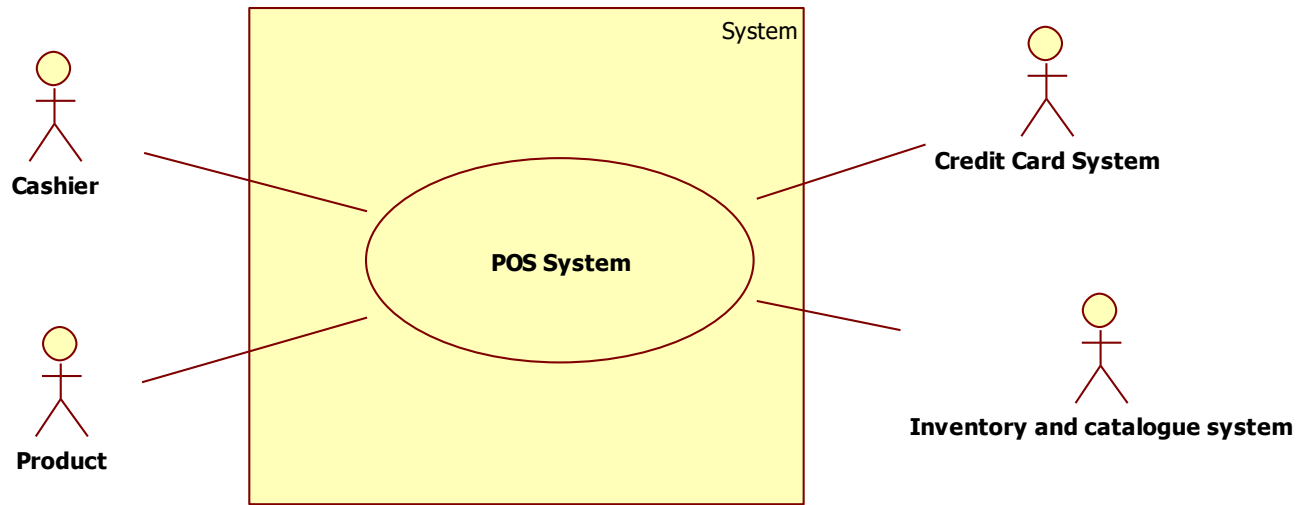  - Exp date incorrect
  - Amount too high
  - …

# System design

- Subsystems (software and not software) that compose the system
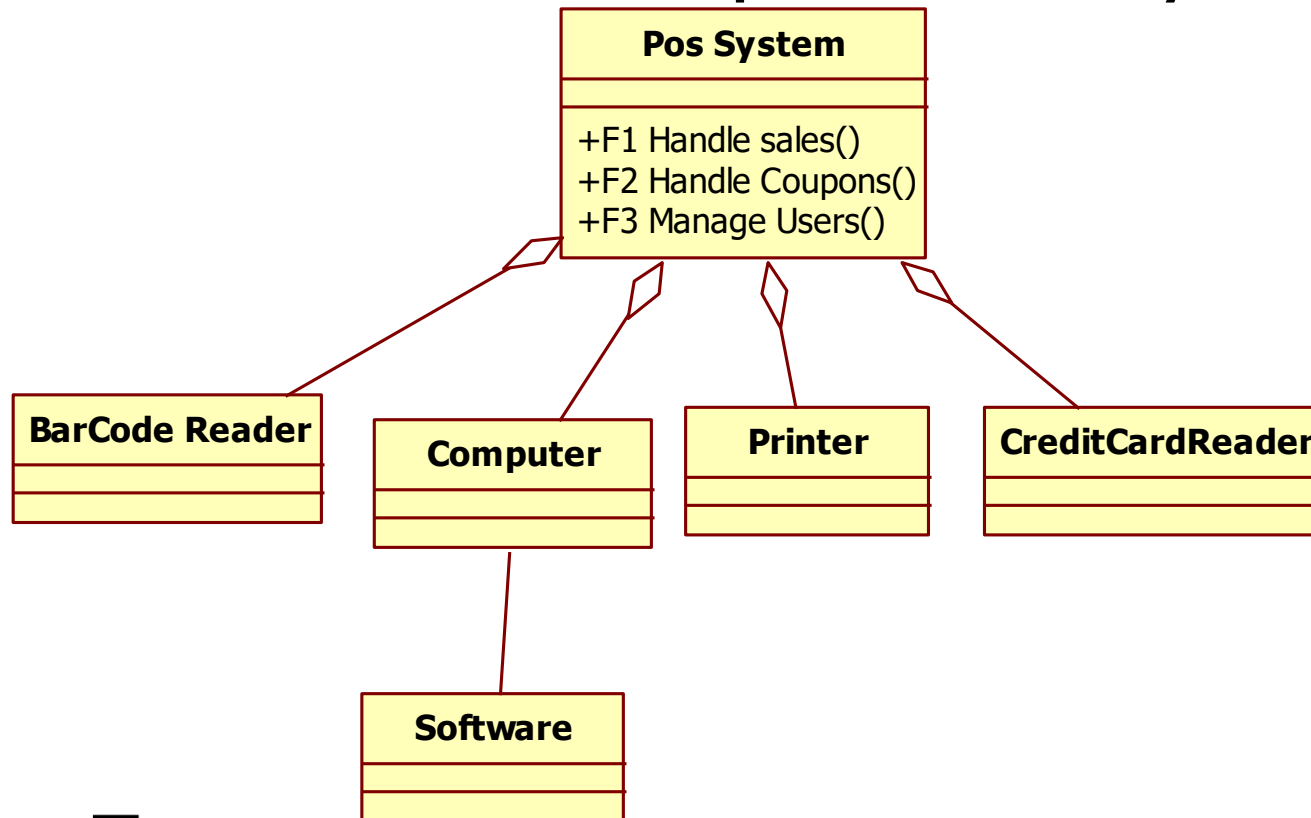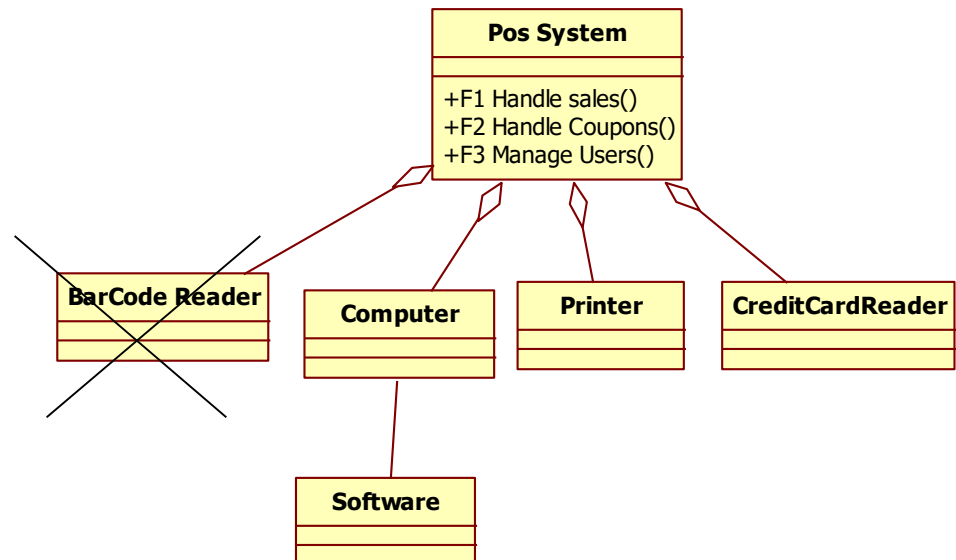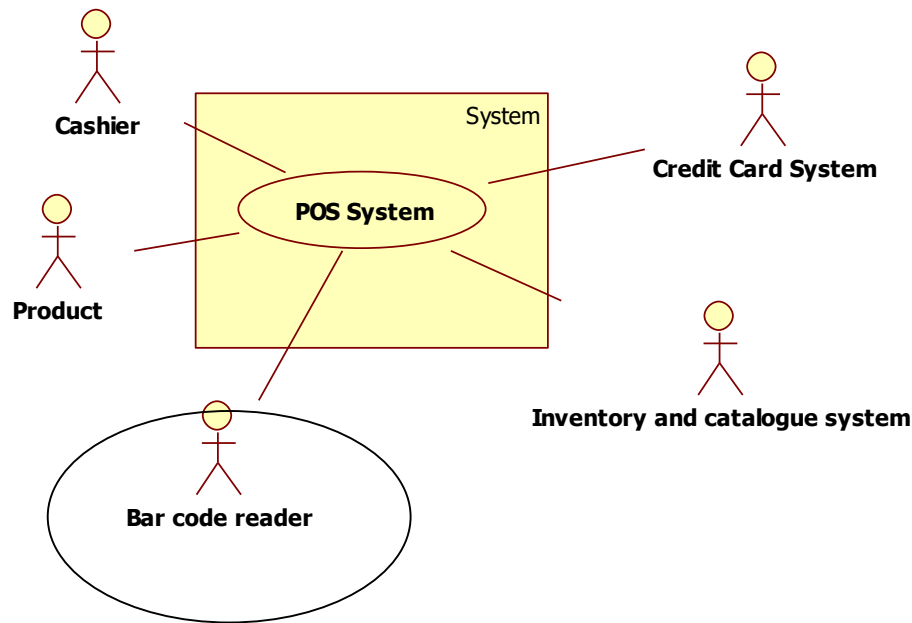
# System design and context diagram

- System design lists all subsystems (or components) that are INSIDE the context diagram

- System design must be consistent with context diagram

■ Bar code reader is IN

**Cashier**

**Product**

System

POS System

**Credit Card System**

**Inventory and catalogue system**

**Bar code reader**

**Pos System**

+F1 Handle sales()
+F2 Handle Coupons()
+F3 Manage Users()

**BarCode Reader**

**Computer**

**Printer**
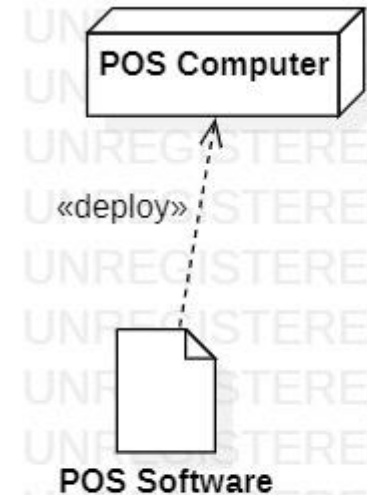
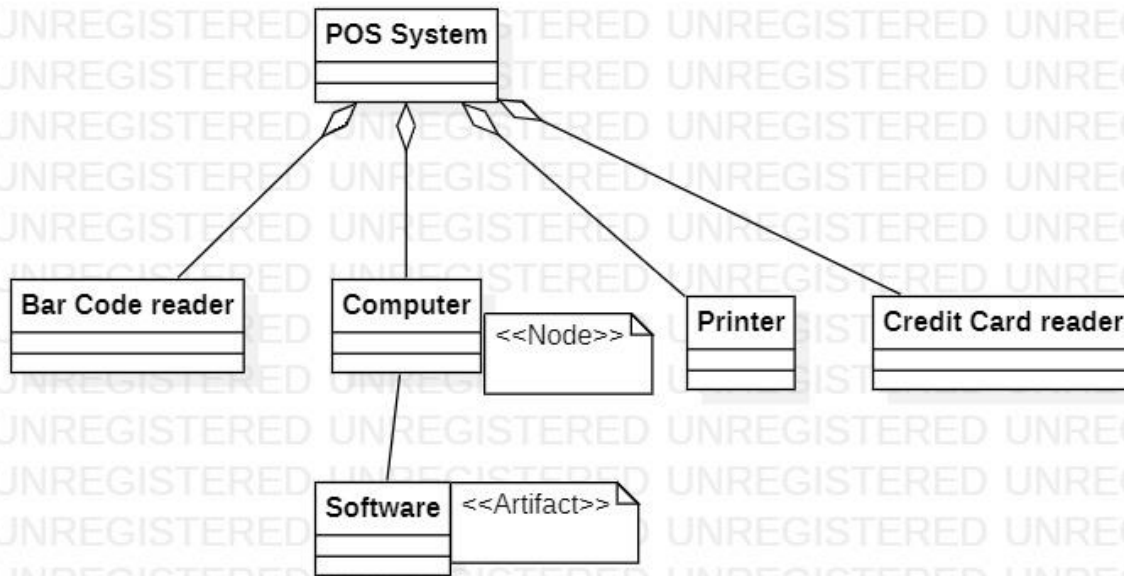**CreditCardReader**

**Software**

- Bar code reader is OUT

# System design vs deployment d

- Class that represents a computing component → Node
- Class that represents a software component → Artifact
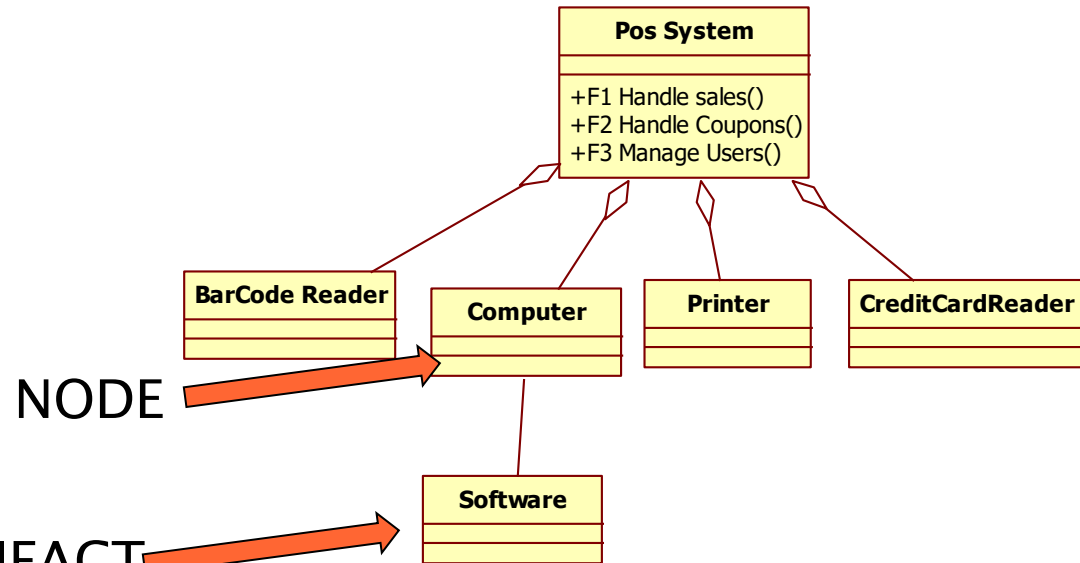
# System design vs deployment

# The requirement document

- Result of RE
- Formalizes the requirements

# Requirement doc structure

1 Overall description

2 Stakeholders

3 Context diagram and interfaces

4 Requirements
- ◆ Functional
- ◆ Non functional
- ◆ Domain

5 Use case diagram
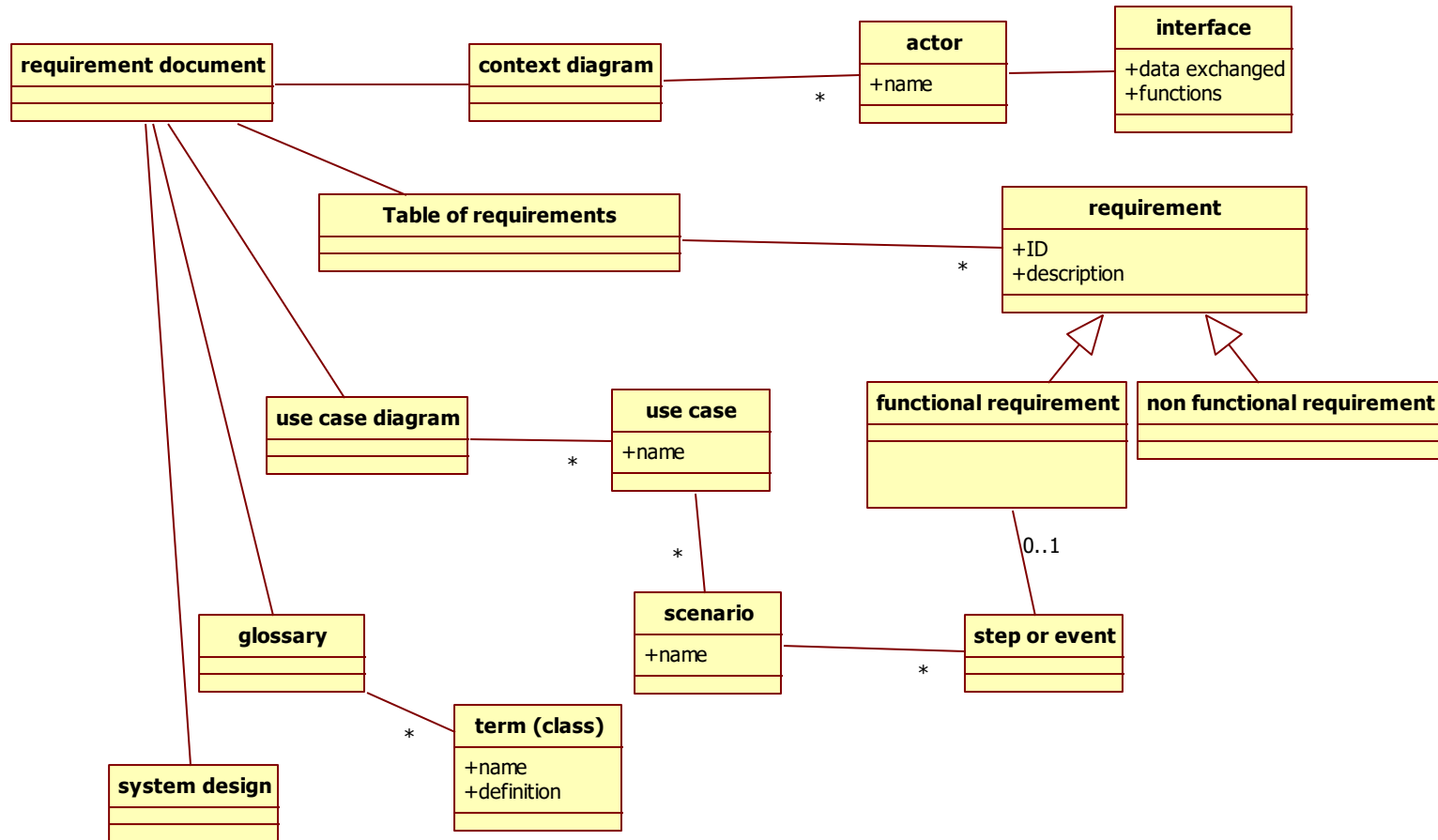
6 Scenarios

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, UC briefs |
| ▪ Scenarios | ▪ Tables, text |
| ▪ Glossary | ▪ Text, UML CD |
| ▪ System design | ▪ UML CD |

# RE – techniques

# Other requirement structures

- http://readyset.tigris.org

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

# Techniques – elicitation

- ◆ Focus group
- ◆ Interviews
- ◆ Questionnaire
- ◆ Ethnography

# Focus group

- Moderator + group of homogeneous people
- Moderator starts and monitors discussion on a defined topics
- Open or script guided

- Long
- Group interaction

# Questionnaire

- Written questions with open /close answers

- Statistical analysis, more data points possible

# Interview

- Deep discussion, one to one
- Open or guided by script / questions


- Interviewer reports log


- Long, detailed

# Ethnographics

- Researcher is 'hidden' in an environment and observes facts and behaviour of user / population

- Expensive, long
- Risk of being invasive

# Styles in RD

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

# Techniques – V and V

- ◆ Inspections / reading
- ◆ Prototyping
- ◆ Iterations
- ◆ (Model checking on formal languages)

# Inspections

- See V&V chapter

# Prototyping

- Develop a running prototype
  - Capable of implementing the key functional requirements
  - Loose on non functional requirements
  - Fast (must be ready in a fraction of the time needed to develop the complete application)
  - Possibly using a dedicated technology, different from the one used for the final system (ex prototype in matlab, application in C)

# Iterations

- Develop a first running version of product, with most important requirements
  - 60 requirements -> 6 months
  - 10 requirements -> 1 month
- To be evolved in final system

  - See process chapter

# MVP

- Minimum viable product
- Develop essential functionality
- Try it with end users

# Support Tools for RE

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

# 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

SoftEng
http://softeng.polito.it

# 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

# 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

**SOftEng**
http://softeng.polito.it

# 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
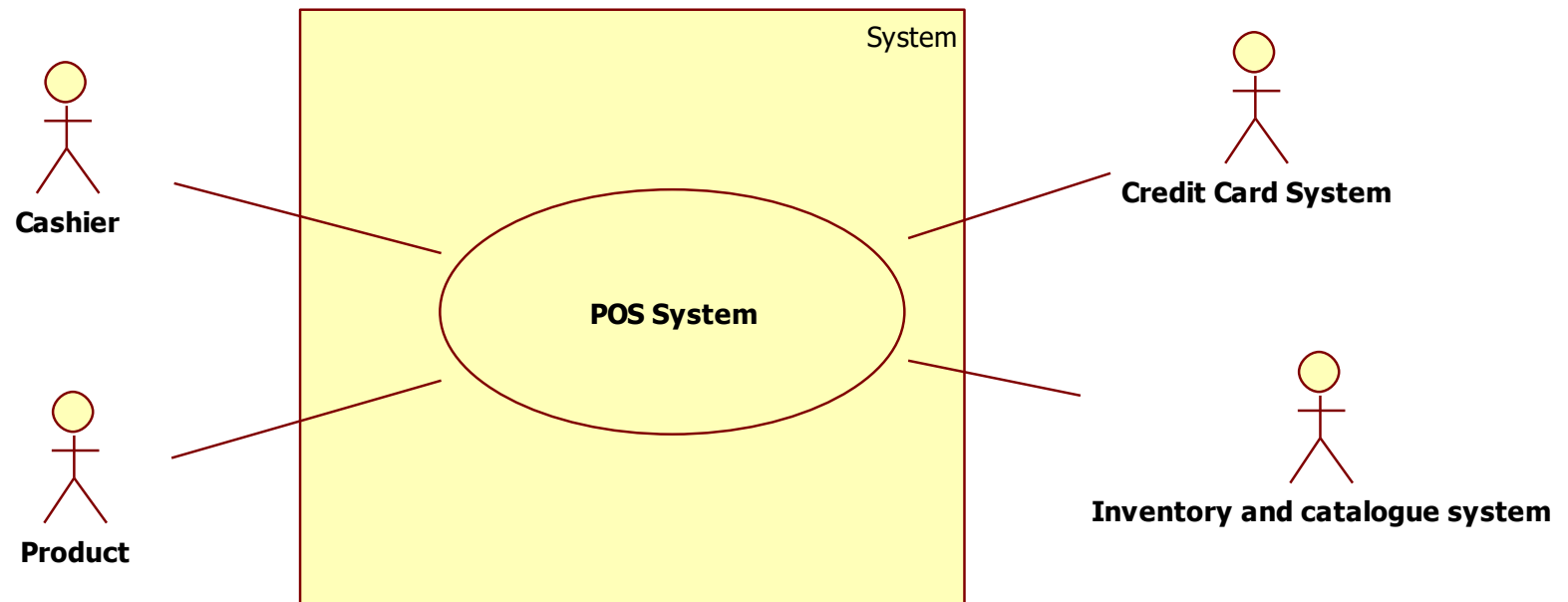
# 3.2 GUI interface

- Sketch of interface, typically built with GUI builder
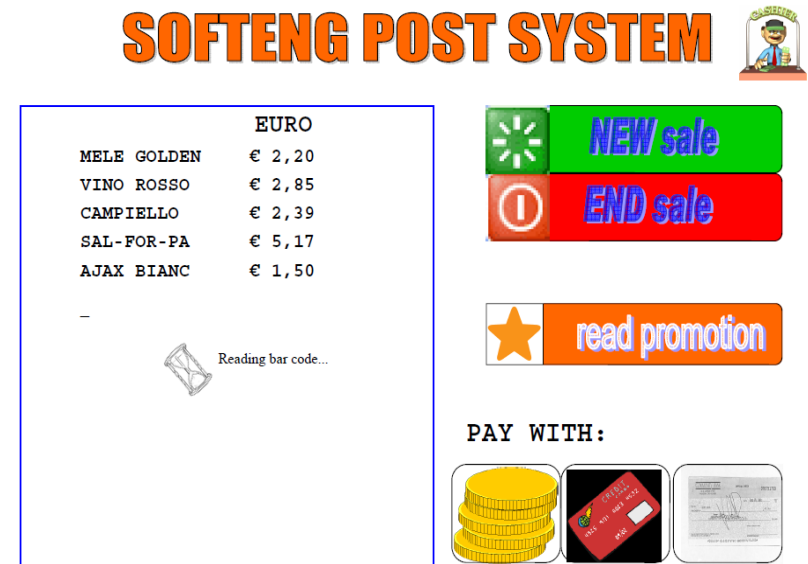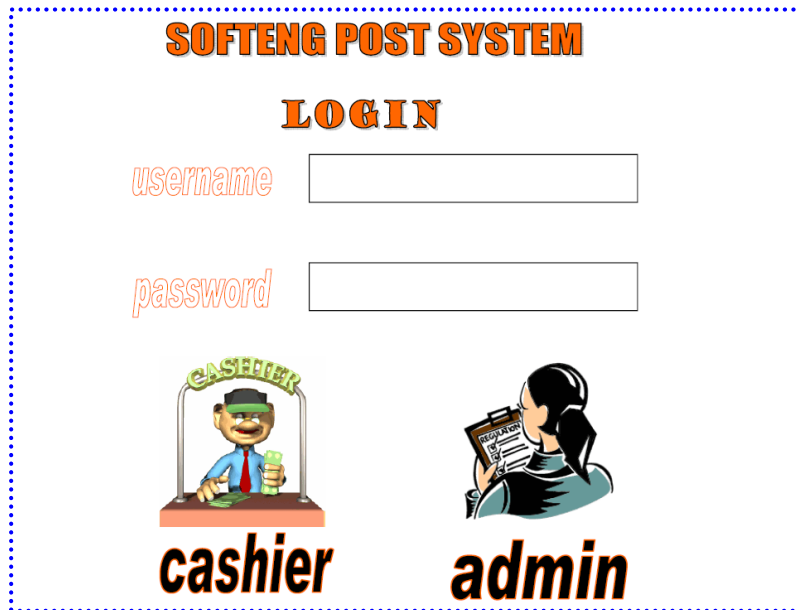
# 4.1 Functional requirements

| Requirement 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
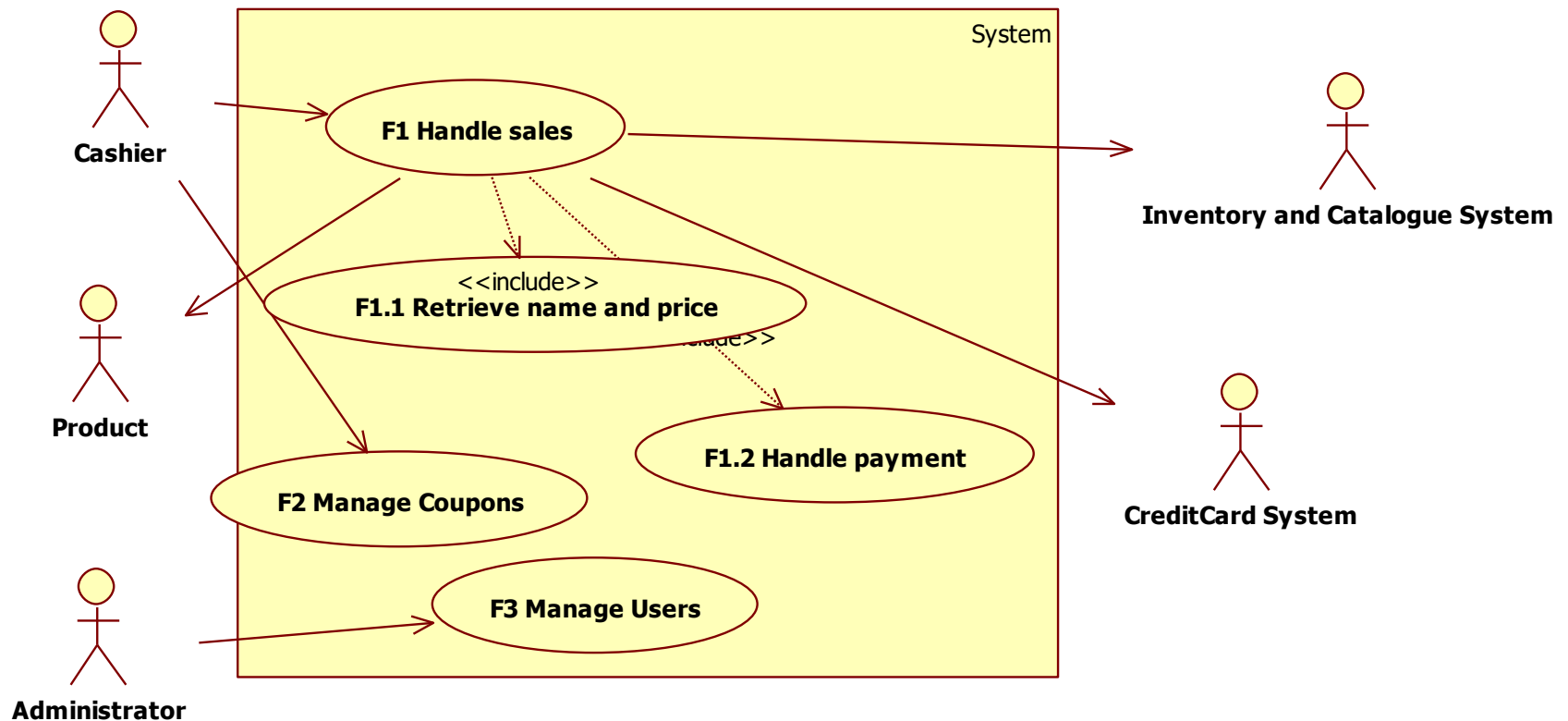
# 6 Scenarios

| 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 |
| |

| Product |
|---|
| +name |
| +barcode |
| |

*

describes

*

SOftEng
http://softeng.polito.it

# 8 System design



**Pos System**

+F1 Handle sales()
+F2 Handle Coupons()
+F3 Manage Users()

**BarCode Reader**

**Computer**

**Printer**

**CreditCardReader**

**Software**