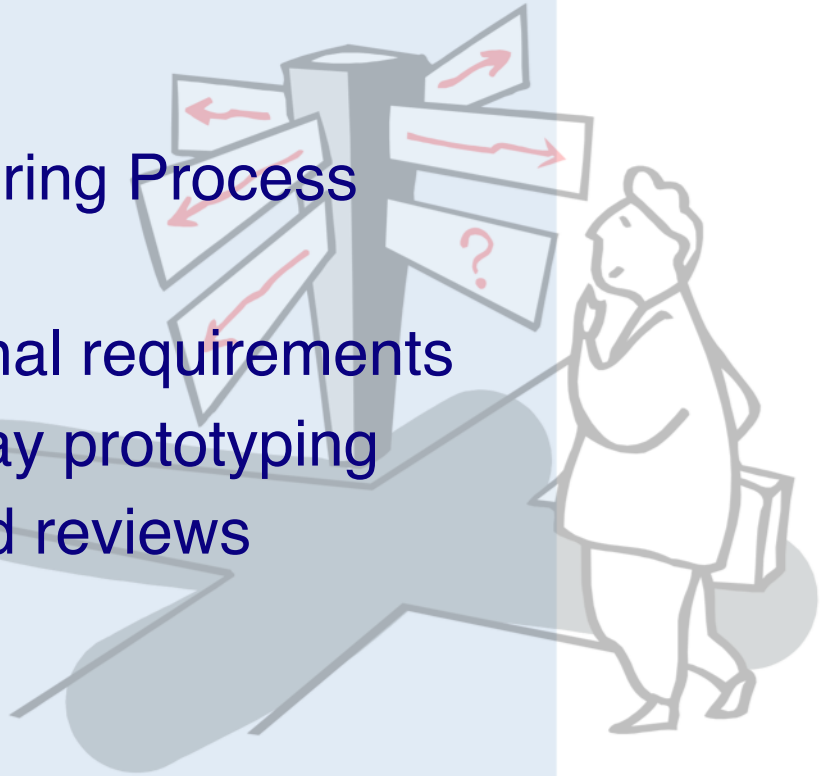
UNIVERSITÄT
BERN

# Introduction to Software Engineering

## 2. Requirements Collection

# Roadmap

- The Requirements Engineering Process
- Use Cases
- Functional and non-functional requirements
- Evolutionary and throw-away prototyping
- Requirements checking and reviews

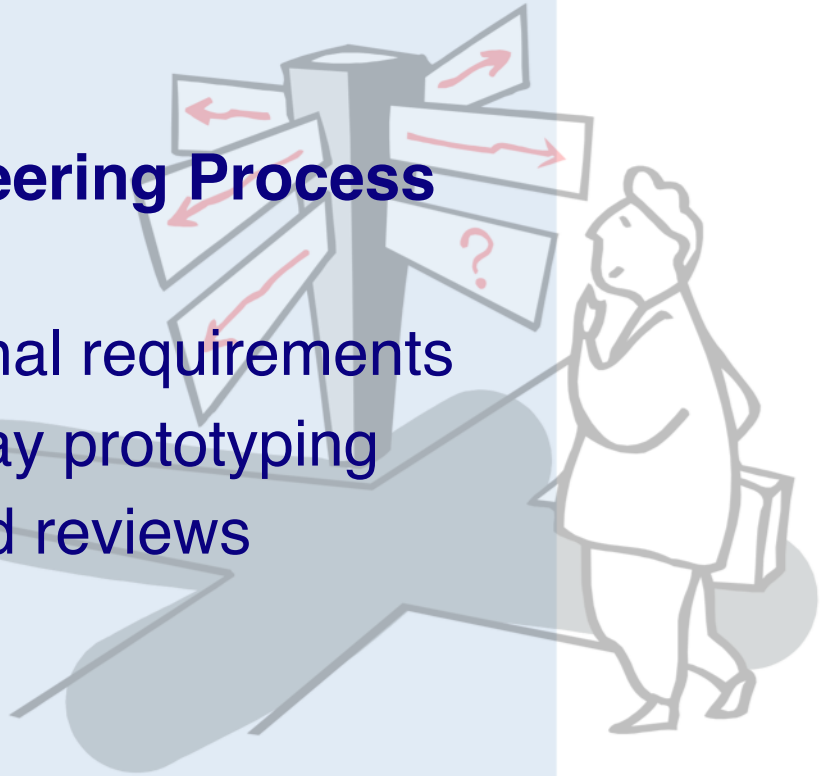


# Sources

- *Software Engineering*, I. Sommerville, 7th Edn., 2004.
- *Software Engineering — A Practitioner's Approach*, R. Pressman, Mc-Graw Hill, 5th Edn., 2001.
- *Objects, Components and Frameworks with UML*, D. D'Souza, A. Wills, Addison-Wesley, 1999

# Roadmap

- **The Requirements Engineering Process**
- Use Cases
- Functional and non-functional requirements
- Evolutionary and throw-away prototyping
- Requirements checking and reviews



**Zeitschema**

Kommission: _____

*Bitte ankreuzen wo Sie keinenfalls mitmachen können, und senden Sie das ausgefüllte Formular bis _____ ans Dekanat zurück.*

| | Jan 2002 | | | | | | | | | | | | | | | | | | | | | | | | Feb 2002 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 8:00 - 9:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9:00 - 10:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10:00 - 11:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 11:00 - 12:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 12:00 - 13:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13:00 - 14:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 14:00 - 15:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 15:00 - 16:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16:00 - 17:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 17:00 - 18:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

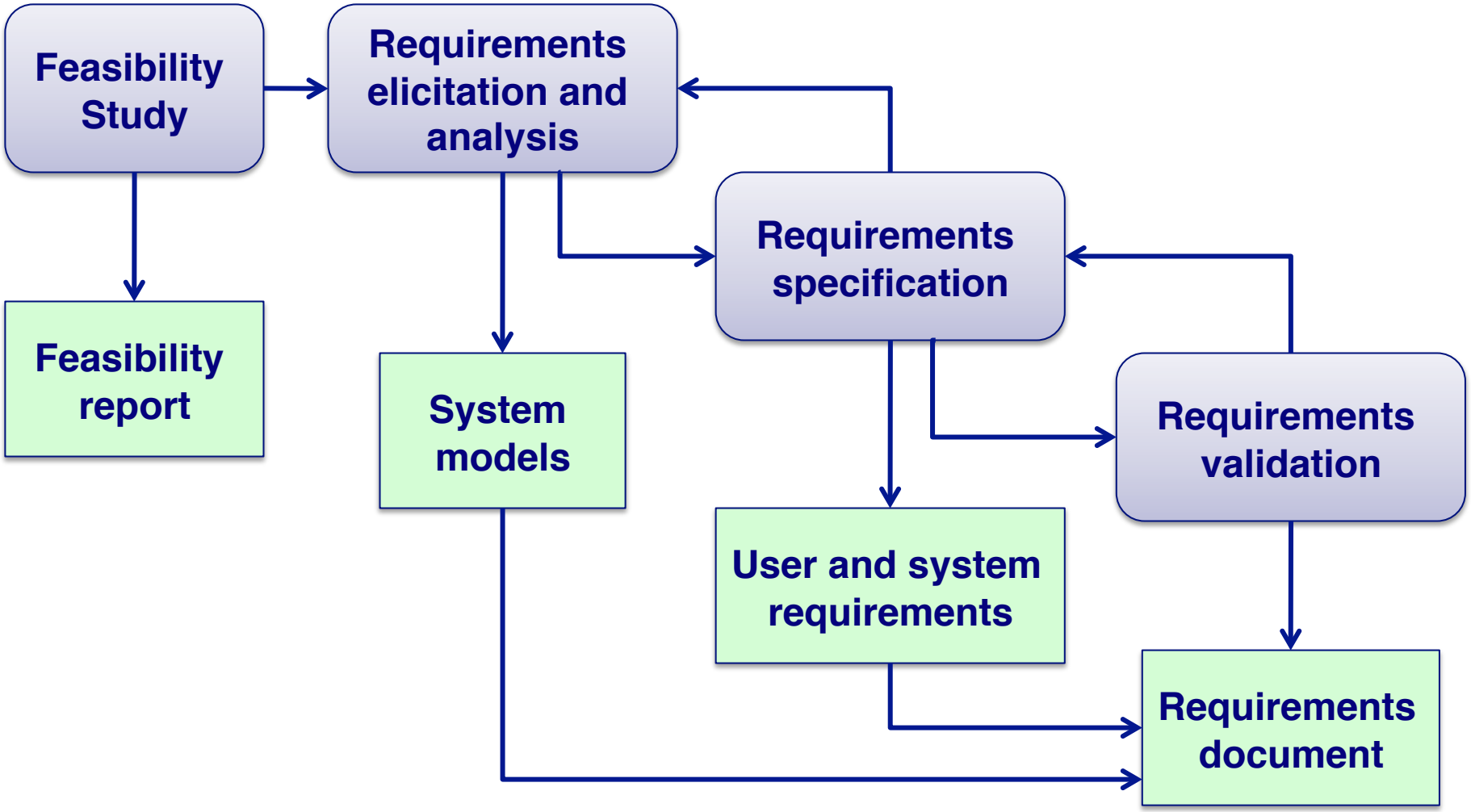
Bemerkungen: _____ Unterschrift: _____

# Electronic Time Schedule

*"So, basically we need a form for the time schedule that can be distributed by eMail, a place (html) where I can deposit these forms after they have been filled out, and an algorithm that calculates a few possible meeting times, possibly setting priorities to certain persons of each committee (since there will always be some time schedule overlaps). It would also be great if there were a way of checking whether everybody of the relevant committee has really sent their time schedule back and at the same time listing all the ones who have failed to do so. An automatic invitation letter for the committee meeting to all the persons involved, generated through this program, would be even a further asset."*

*How can we transform this description into a requirements specification?*

# The Requirements Engineering Process



# Requirements Engineering Activities

| | |
|---|---|
| ***Feasibility study*** | Determine if the *user needs* can be *satisfied* with the *available technology* and *budget*. |
| ***Requirements analysis*** | Find out *what system stakeholders require* from the system. |
| ***Requirements definition*** | *Define the requirements* in a form understandable to the customer. |
| ***Requirements specification*** | Define the requirements in *detail*. (Written as a contract between client and contractor.) |

*"Requirements are for users; specifications are for analysts and developers."*

# Requirements Analysis

Sometimes called *requirements elicitation* or *requirements discovery*

Technical staff work with customers to determine

- the application *domain*,
- the *services* that the system should provide and
- the system's operational *constraints*.

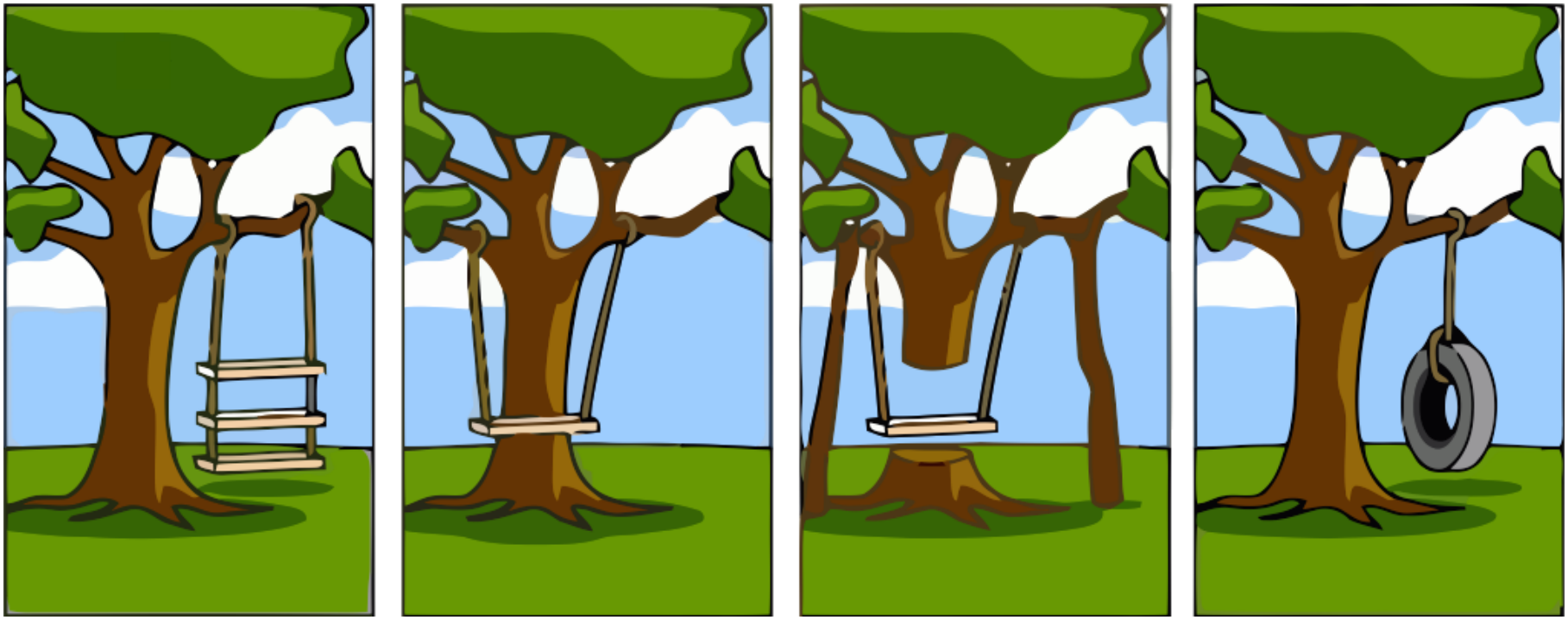
Involves various *stakeholders*:

- e.g., end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc.

# Problems of Requirements Analysis

Various problems typically arise:

- Stakeholders *don't know* what they really want
- Stakeholders express requirements *in their own terms*
- Different stakeholders may have *conflicting requirements*
- *Organisational and political factors* may influence the system requirements
- The *requirements change* during the analysis process.
- *New stakeholders* may emerge.



**How the Customer explained it**

**How the Project Leader understood it**

**How the Analyst designed it**

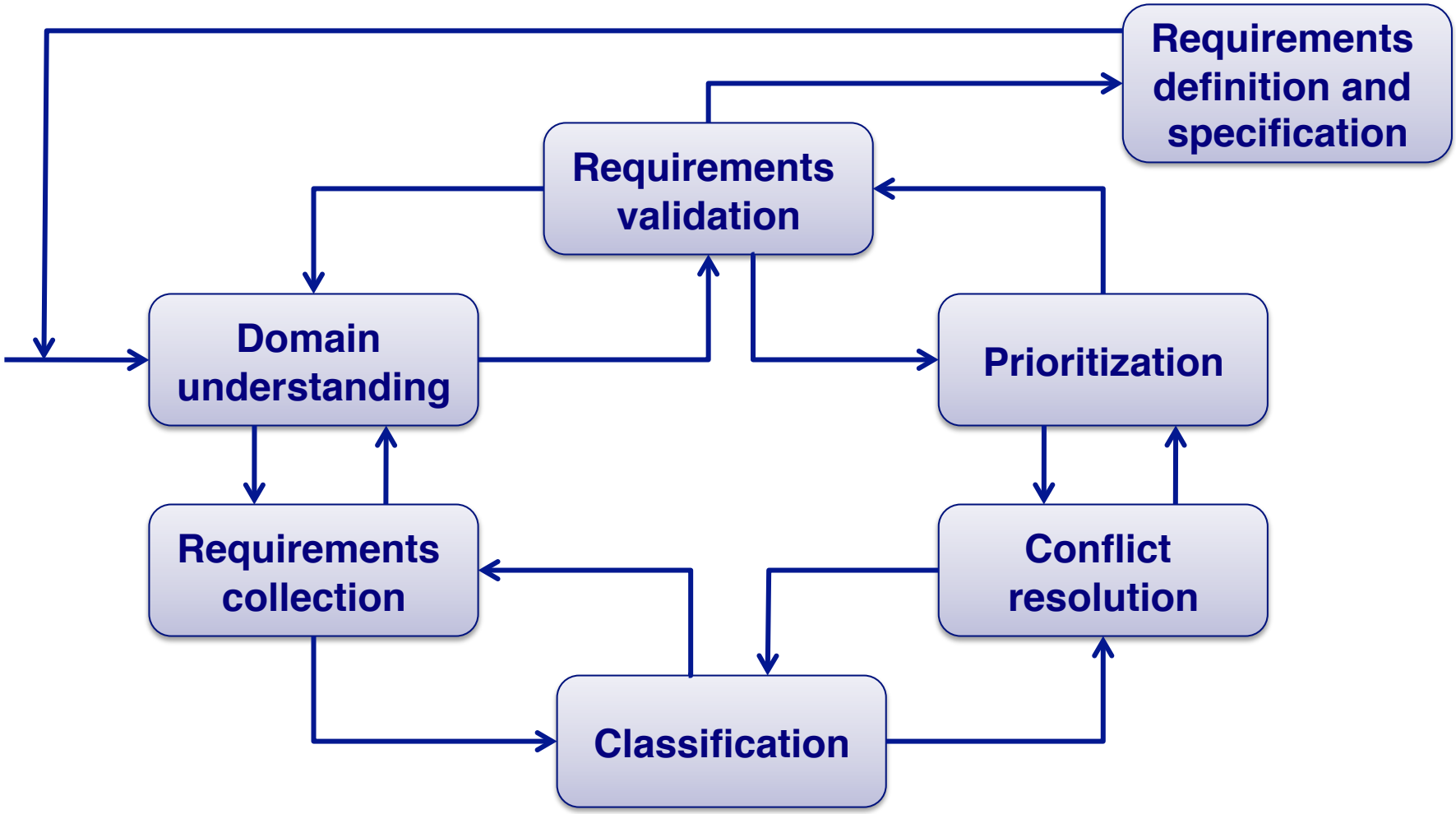
**What the Customer really needed**

# Requirements evolution

- Requirements *always evolve* as a better understanding of user needs is developed and as the organisation's objectives change

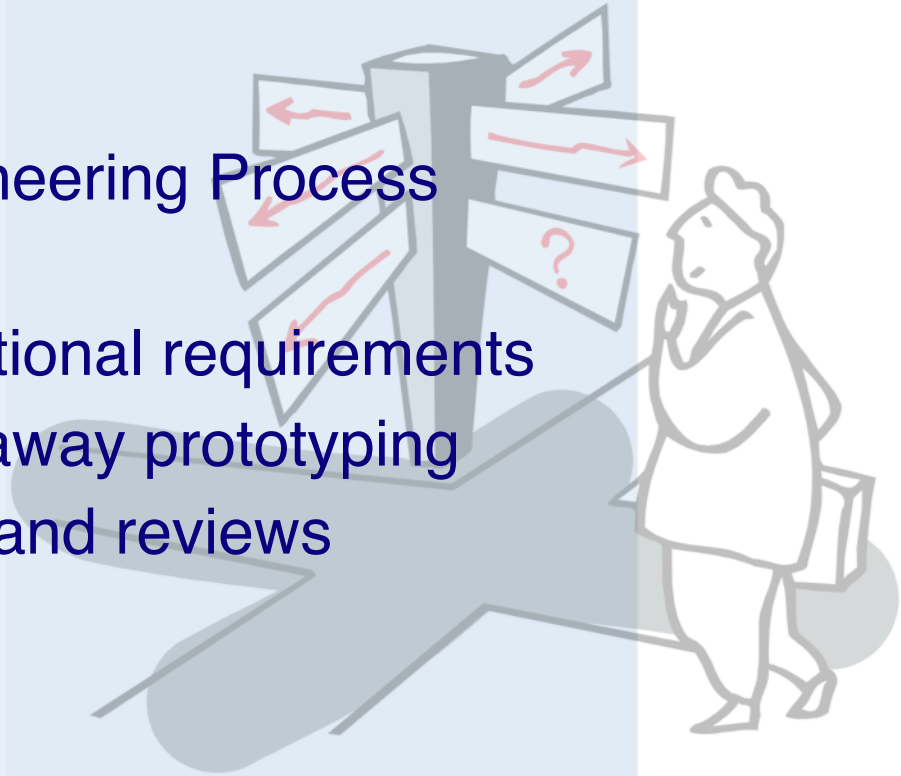
- It is essential to *plan for change* in the requirements as the system is being developed and used

# The Requirements Analysis Process



# Roadmap

- The Requirements Engineering Process
- **Use Cases**
- Functional and non-functional requirements
- Evolutionary and throw-away prototyping
- Requirements checking and reviews



# Use Cases and Scenarios

A *use case* is the *specification* of a *sequence of actions*, including *variants*, that a system (or other entity) can perform, *interacting with actors* of the system".

— e.g., buy a DVD through the internet

A *scenario* is a *particular trace of action occurrences*, starting from a known initial state.

— e.g., connect to myDVD.com, go to the "search" page

...

# Use Cases and Viewpoints ...

**Stakeholders** represent different problem *viewpoints*.

- Interview as many *different* kinds of stakeholders as possible/ necessary
- Translate requirements into *use cases* or “stories” about the desired system involving a fixed set of actors (users and system objects)
- For each use case, capture *both typical and exceptional* usage scenarios

**Users** tend to think about systems in terms of “features”.

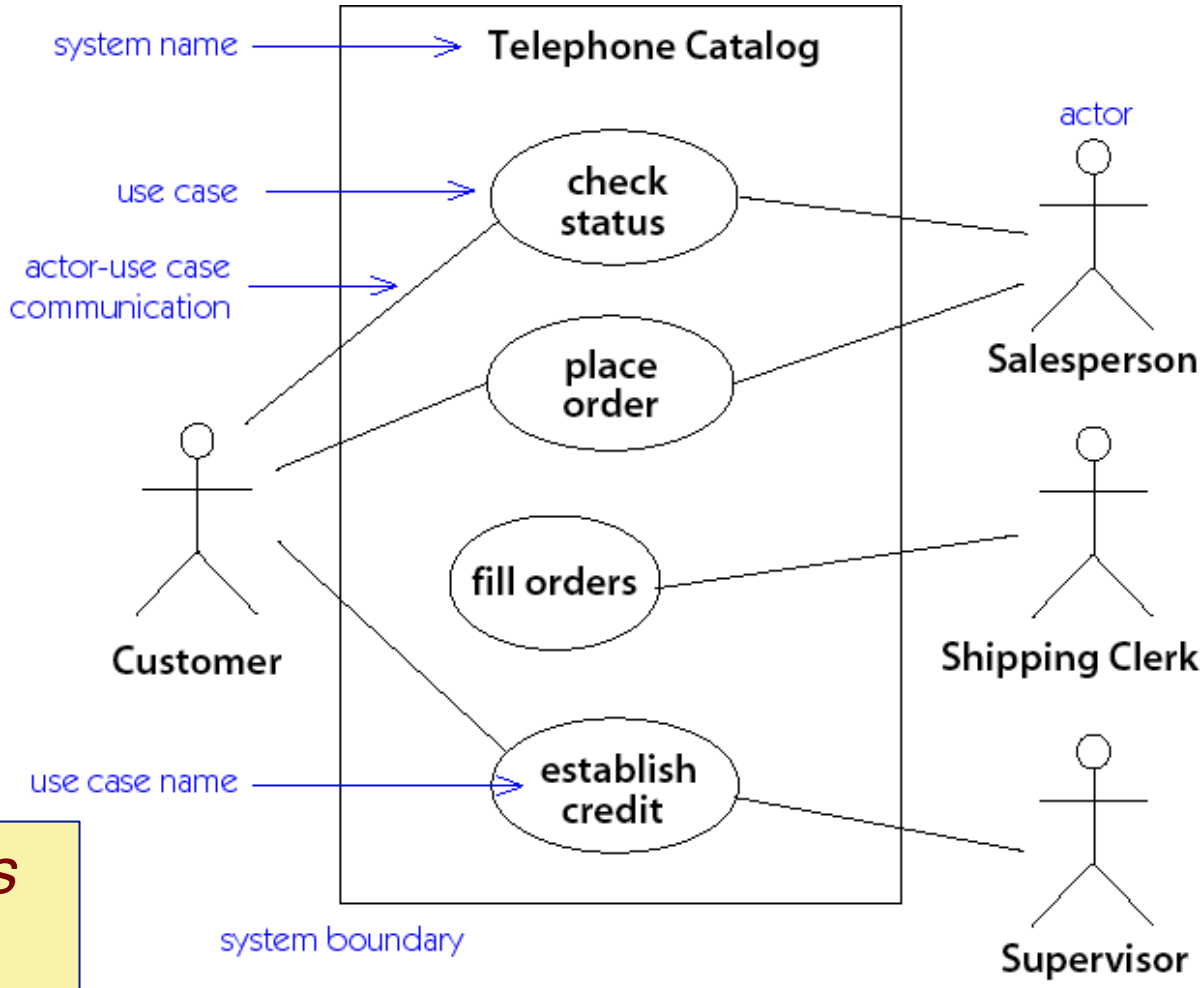
- You must get them to tell you *stories* involving those features.
- Use cases and scenarios can tell you if the requirements are *complete and consistent*!

# Unified Modeling Language

*UML is the industry standard for documenting OO models*

| | |
|---|---|
| ***Class Diagrams*** | visualize *logical structure* of system<br>in terms of *classes, objects and relationships* |
| ***Use Case Diagrams*** | show external *actors and use cases* they participate in |
| ***Sequence Diagrams*** | visualize *temporal message ordering* of a *concrete scenario* of a use case |
| ***Collaboration (Communication) Diagrams*** | visualize *relationships* of objects exchanging messages in a *concrete scenario* |
| ***State Diagrams*** | specify the *abstract states* of an object and the *transitions* between the states |

# Use Case Diagrams




*More on this later …*

**Figure 5-1.** *Use case diagram*

# Sequence Diagrams

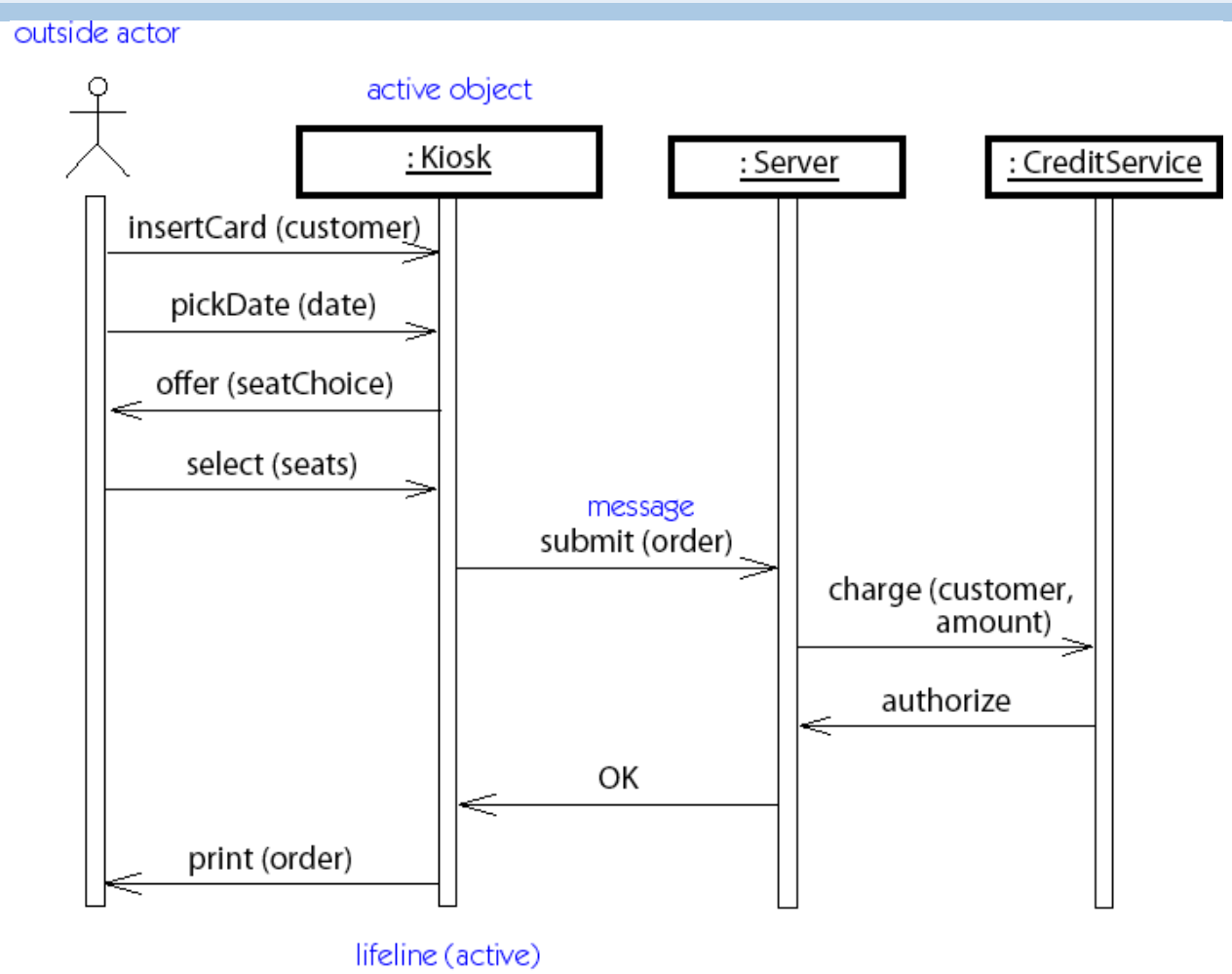



**Figure 8-1.** *Sequence diagram*

# Writing Requirements Definitions

Requirements definitions usually consist of *natural language*, supplemented by (e.g., UML) *diagrams and tables*.

*Three types of problems can arise:*

- **Lack of clarity**: It is hard to write documents that are both *precise and easy-to-read*.
- **Requirements confusion**: *Functional and non-functional requirements* tend to be intertwined.
- **Requirements amalgamation**: Several *different requirements* may be expressed together.

# Roadmap

- The Requirements Engineering Process
- Use Cases
- **Functional and non-functional requirements**
- Evolutionary and throw-away prototyping
- Requirements checking and reviews

# Functional and Non-functional Requirements

*Functional requirements* describe system *services* or *functions*

- Compute sales tax on a purchase
- Update the database on the server ...

*Non-functional requirements* are *constraints* on the system or the development process

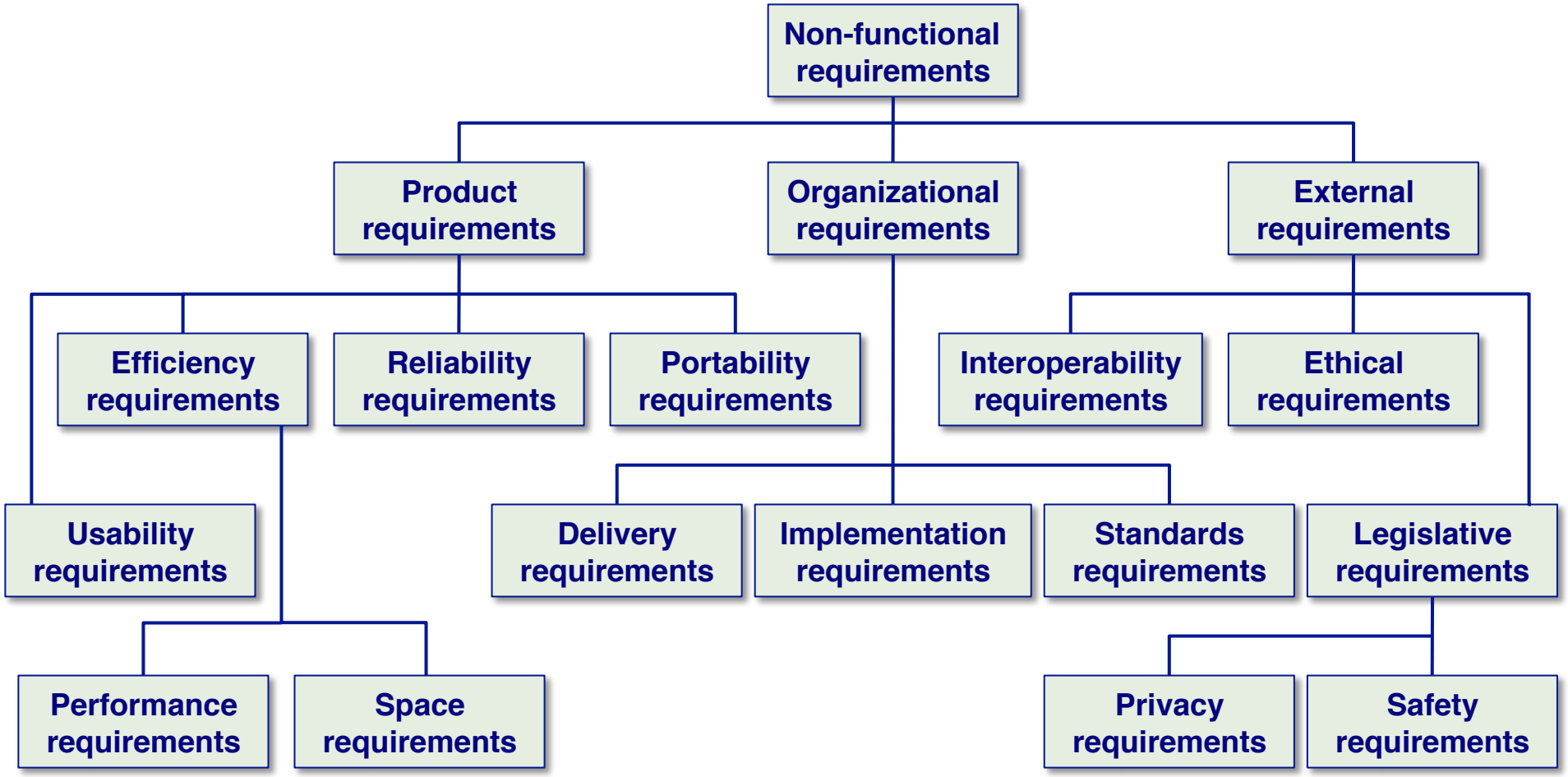
*Non-functional requirements may be more critical than functional requirements.*

*If these are not met, the system is useless!*

# Non-functional Requirements

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

# Types of Non-functional Requirements



# Examples of Non-functional Requirements

| | |
|---|---|
| ***Product requirement*** | It shall be possible for all necessary communication between the APSE and the user to be expressed in the *standard Ada character set.* |
| ***Organisational requirement*** | The *system development process* and deliverable documents shall conform to the process and deliverables defined in *XYZCo-SP-STAN-95.* |
| ***External requirement*** | The system shall provide facilities that allow any user to check if personal data is maintained on the system. *A procedure must be defined and supported in the software that will allow users to inspect personal data* and to correct any errors in that data. |

# Requirements Verifiability

Requirements must be written so that they can be *objectively verified*.

***Imprecise:***

— The system should be *easy to use* by experienced controllers and should be organised in such a way that *user errors are minimised.*

*Terms like "easy to use" and "errors shall be minimised" are useless as specifications.*

***Verifiable:***

— Experienced controllers should 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 should not exceed two per day*.

# Precise Requirements Measures (I)

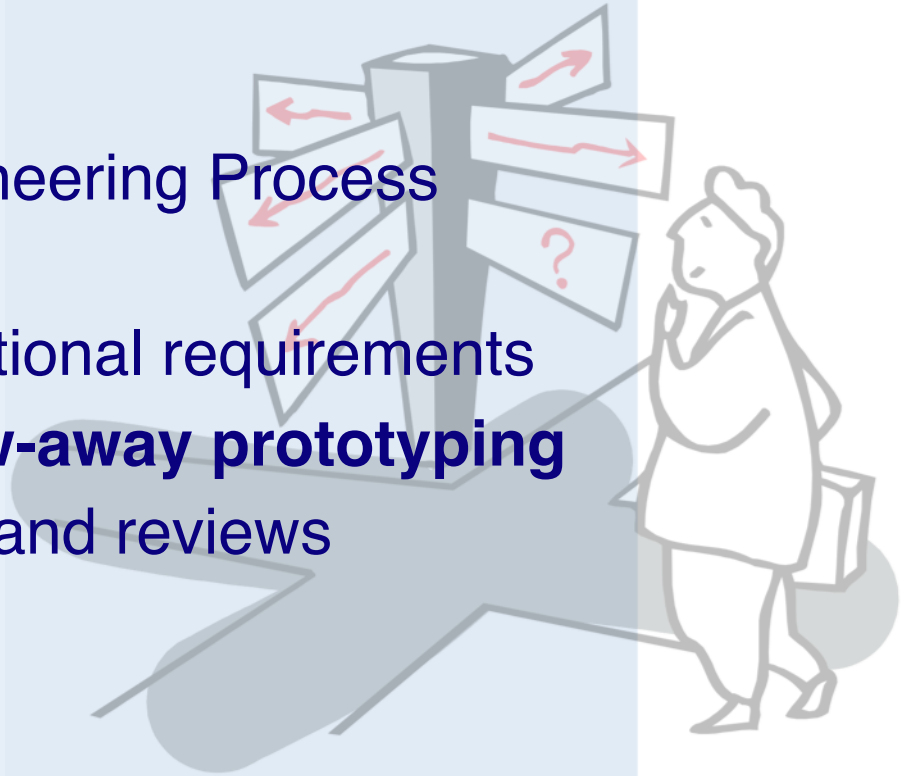
| ***Property*** | ***Measure*** |
|---|---|
| *Speed* | Processed transactions/second<br>User/Event response time<br>Screen refresh time |
| *Size* | K Bytes; Number of RAM chips |
| *Ease of use* | Training time<br>Rate of errors made by trained users<br>Number of help frames |

# Precise Requirements Measures (II)

| ***Property*** | ***Measure*** |
|---|---|
| *Reliability* | Mean time to failure<br>Probability of unavailability<br>Rate of failure occurrence |
| *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 |

# Roadmap

- The Requirements Engineering Process
- Use Cases
- Functional and non-functional requirements
- **Evolutionary and throw-away prototyping**
- Requirements checking and reviews



# Prototyping Objectives

The objective of *evolutionary prototyping* is to deliver a *working system* to end-users.

— Development starts with the requirements that are *best understood*.

The objective of *throw-away prototyping* is to *validate or derive the system requirements*.

— Prototyping starts with that requirements that are *poorly understood*.

# Evolutionary Prototyping

- Must be used for systems where *the specification cannot be developed in advance*.
  - *e.g., AI systems and user interface systems*
- Based on techniques which allow *rapid system iterations*.
  - *e.g., executable specification languages, VHL languages, 4GLs, component toolkits*
- *Verification is impossible* as there is no specification.
  - *Validation means demonstrating the adequacy of the system.*

# Throw-away Prototyping

- Used to *reduce requirements risk*
  - The prototype is *developed* from an initial specification, *delivered* for experiment then *discarded*
- The throw-away prototype should *not* be considered as a final system
  - Some system characteristics may have been left out
  - (e.g., platform requirements may be ignored)
  - There is no specification for long-term maintenance
  - The system will be poorly structured and difficult to maintain

# Roadmap

- The Requirements Engineering Process
- Use Cases
- Functional and non-functional requirements
- Evolutionary and throw-away prototyping
- **Requirements checking and reviews**

# Requirements Checking

| | |
|---|---|
| ***Validity*** | Does the system provide the functions *which best support* the customer's needs? |
| ***Consistency*** | Are there any *requirements conflicts*? |
| ***Completeness*** | Are *all functions* required by the customer included? |
| ***Realism*** | Can the requirements be implemented given *available budget and technology*? |

# Requirements Reviews

- *Regular reviews* should be held while the requirements definition is being formulated
- Both *client and contractor* staff should be involved in reviews
- Reviews may be *formal* (with completed documents) or *informal*.
  - *Good communications* between developers, customers and users can resolve problems at an *early stage*

# Review checks

| | |
|---|---|
| ***Verifiability*** | Is the requirement realistically *testable*? |
| ***Comprehensibility*** | Is the requirement properly *understood*? |
| ***Traceability*** | Is the *origin* of the requirement clearly stated? |
| ***Adaptability*** | Can the requirement be *changed* without a large *impact* on other requirements? |

# Sample Requirements Review Checklist

- Does the (software) product have a *succinct name*, and a *clearly described purpose*?
- Are the *characteristics of users* and of *typical usage* mentioned?
  (No user categories missing.)
- Are all *external interfaces* of the software explicitly mentioned?
  (No interfaces missing.)
- Does each specific requirement have a *unique identifier* ?
- Is each requirement *atomic* and *simply formulated* ?
  (Typically a single sentence. Composite requirements must be split.)
- Are requirements organized into *coherent groups* ?
  (If necessary, hierarchical; not more than about ten per group.)
- Is each requirement *prioritized* ?
  (Is the meaning of the priority levels clear?)
- Are all *unstable requirements* marked as such?
  (TBC=`To Be Confirmed', TBD=`To Be Defined')

http://wwwis.win.tue.nl/2M390/rev_req.html

# Sample Requirements Review Checklist

- Is each requirement *verifiable* (in a provisional acceptance test)? (Measurable: where possible, quantify; capacity, performance, accuracy)
- Are the requirements *consistent* ? (Non-conflicting.)
- Are the requirements sufficiently *precise* and *unambiguous* ? (Which interfaces are involved, who has the initiative, who supplies what data, no passive voice.)
- Are the requirements *complete*? Can everything not explicitly constrained indeed be viewed as developer freedom? Is a product that satisfies every requirement indeed acceptable? (No requirements missing.)
- Are the requirements *understandable* to those who will need to work with them later?
- Are the requirements *realizable* within budget?
- Do the requirements express actual *customer needs* (in the language of the problem domain), *rather than solutions* (in developer jargon)?

http://wwwis.win.tue.nl/2M390/rev_req.html

# Traceability

To protect against changes you should be able to *trace back from every system component to the original requirement* that caused its presence

| | *C1* | *C2* | *…* | *…* | *Cm* |
|---|---|---|---|---|---|
| *req1* | | x | | | |
| *req2* | x | | | | |
| *…* | | | | x | |
| *…* | | | | | x |
| *reqn* | | x | x | | |

- *A software process* should help you keep this virtual table up-to-date
- *Simple techniques* may be quite valuable (naming conventions, ...)

# What you should know!

- What is the difference between requirements analysis and specification?
- Why is it hard to define and specify requirements?
- What are use cases and scenarios?
- What is the difference between functional and non-functional requirements?
- What's wrong with a requirement that says a product should be "user-friendly"?
- What's the difference between evolutionary and throw-away prototyping?

# Can you answer the following questions?

- Why isn't it enough to specify requirements as a set of desired features?
- Which is better for specifying requirements: natural language or diagrams?
- How would you prototype a user interface for a web-based ordering system?
- Would it be an evolutionary or throw-away prototype?
- What would you expect to gain from the prototype?
- How would you check a requirement for "adaptability"?

# License