# Chapter 4 – Requirements Engineering

SE2021 SPRING

## Topics covered

- Requirements engineering intro
- How to write good requirements
- Requirements specification document
- Requirements engineering processes
  - Elicitation
  - Analysis
  - Validation
  - evolution

## Requirements engineering

- ✧The process of establishing the *services* that the customer requires from a system and the *constraints* under which it operates and is developed
- ✧The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process

# Definition of RE

**Requirements Engineering (RE)** is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the contexts in which it will be used. Hence, RE acts as the bridge between the real world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies

Not a phase or stage!

Communication is as important as the analysis

Quality means fitness-for-purpose. Cannot say anything about quality unless you understand the purpose

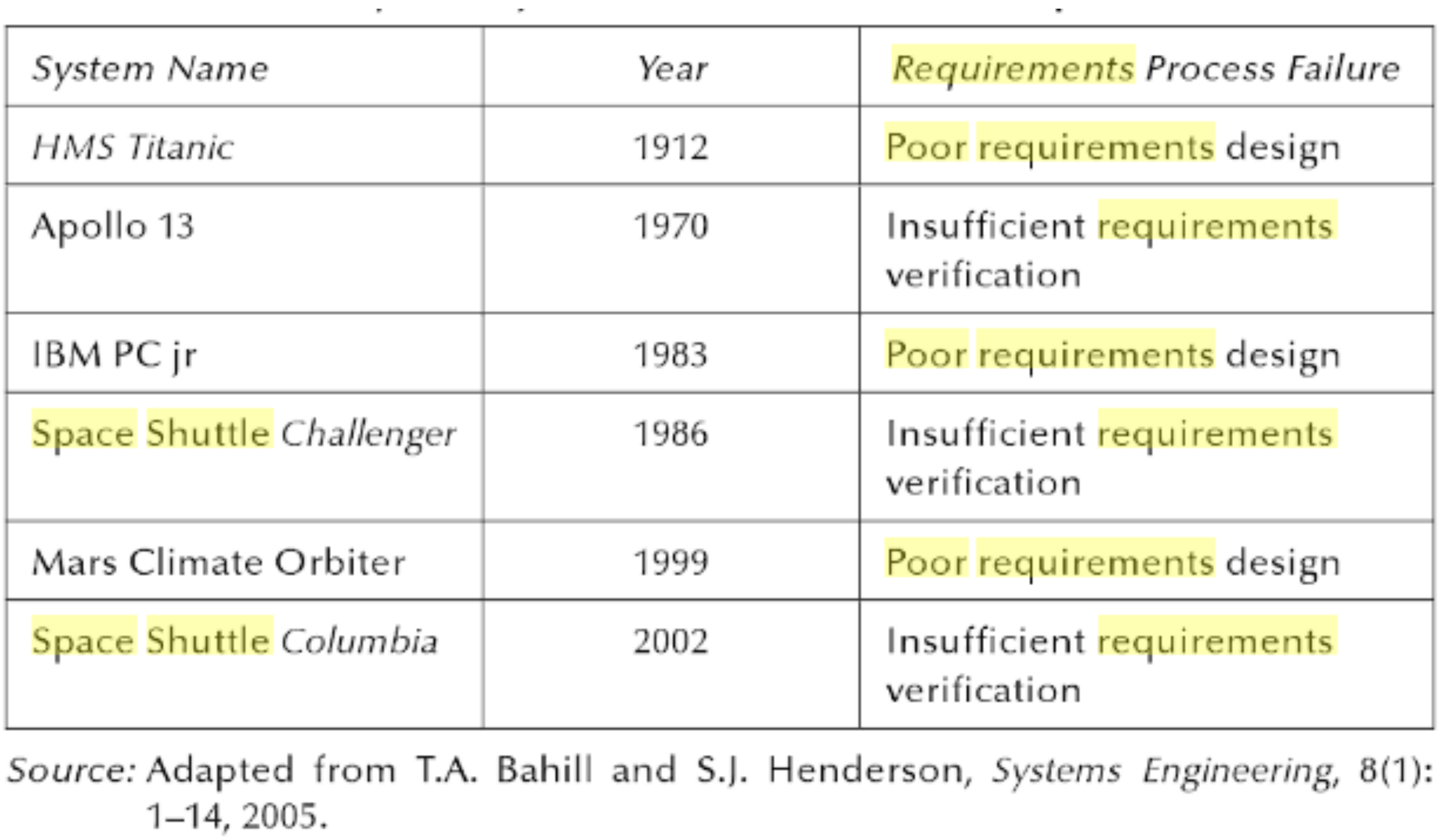
Designers need to know how and where the system will be used

Requirements are partly about what is needed…

…and partly about what is possible

Need to identify all the stakeholders - not just the customer and user

## Why we need RE?



| *System Name* | *Year* | *Requirements Process Failure* |
|---|---|---|
| *HMS Titanic* | 1912 | Poor requirements design |
| Apollo 13 | 1970 | Insufficient requirements verification |
| IBM PC jr | 1983 | Poor requirements design |
| Space Shuttle *Challenger* | 1986 | Insufficient requirements verification |
| Mars Climate Orbiter | 1999 | Poor requirements design |
| Space Shuttle *Columbia* | 2002 | Insufficient requirements verification |

*Source:* Adapted from T.A. Bahill and S.J. Henderson, *Systems Engineering,* 8(1): 1–14, 2005.

## Types of requirement

- ✧User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- ✧System requirements
  - A structured document setting out detailed descriptions of the system’s functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

# User and system requirements

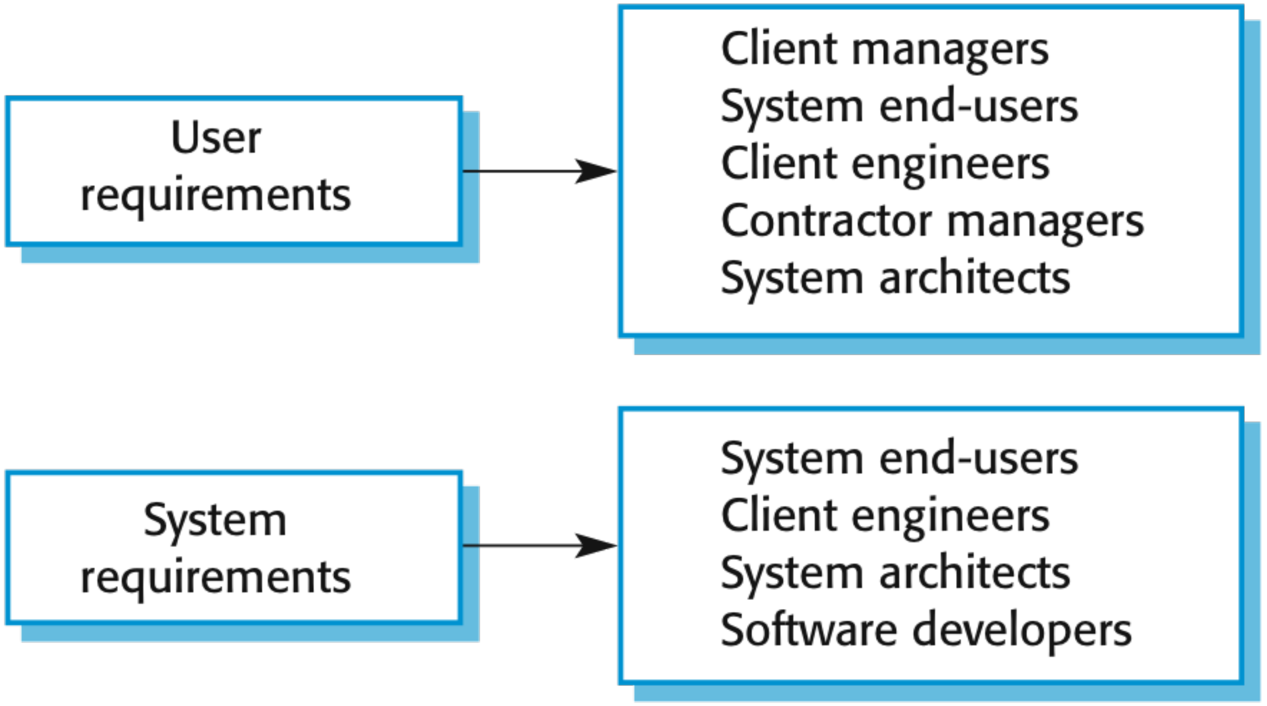
**User requirement definition**

**1.** The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

**System requirements specification**

**1.1** On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
**1.2** The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
**1.3** A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
**1.4** If drugs are available in different dose units (e.g. 10mg, 20 mg, etc.) separate reports shall be created for each dose unit.
**1.5** Access to all cost reports shall be restricted to authorized users listed on a management access control list.

## Readers of different types of requirements specification



## Functional and non-functional requirements

- ✧Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations
  - May state what the system should not do
- ✧Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services
- ✧Domain requirements
  - Constraints on the system from the domain of operation

## Functional requirements

- ✧Describe *functionality* or system *services*
- ✧Depend on the type of software, expected users and the type of system where the software is used
- ✧Functional user requirements may be high-level statements of what the system should do
- ✧Functional system requirements should describe the system services in detail

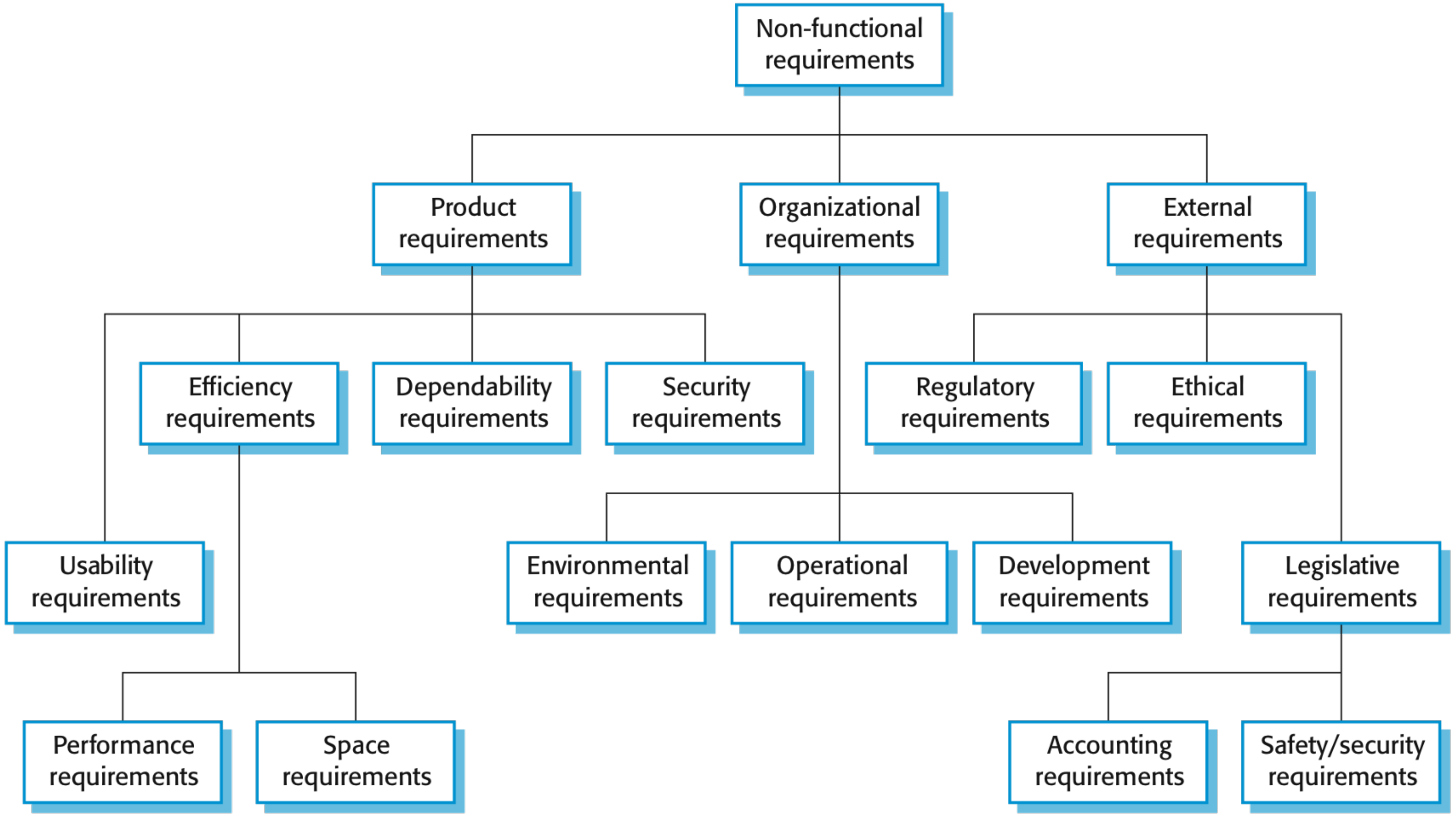
## Functional requirements for the MHC-PMS

- ✧A user shall be able to search the appointments lists for all clinics
- ✧The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day
- ✧Each staff member using the system shall be uniquely identified by his or her 8-digit employee number

## Non-functional requirements

- ✧These define *system properties* and *constraints* e.g. reliability, response time, and storage requirements. Constraints are I/O device capability, system representations, etc.
- ✧Process requirements may also be specified mandating a particular IDE, programming language or development method.
- ✧Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

## Types of nonfunctional requirement



## Non-functional requirements implementation

- ✧Non-functional requirements may affect the overall architecture of a system rather than the individual components
  - For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components
- ✧A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required
  - It may also generate requirements that restrict existing requirements

## Non-functional classifications

- ✧Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- ✧Organizational requirements
  - Requirements which are a consequence of organizational 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.

## Examples of nonfunctional requirements in the MHC-PMS

**Product requirement**
The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

**Organizational requirement**
Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

**External requirement**
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

## Metrics for specifying non-functional requirements

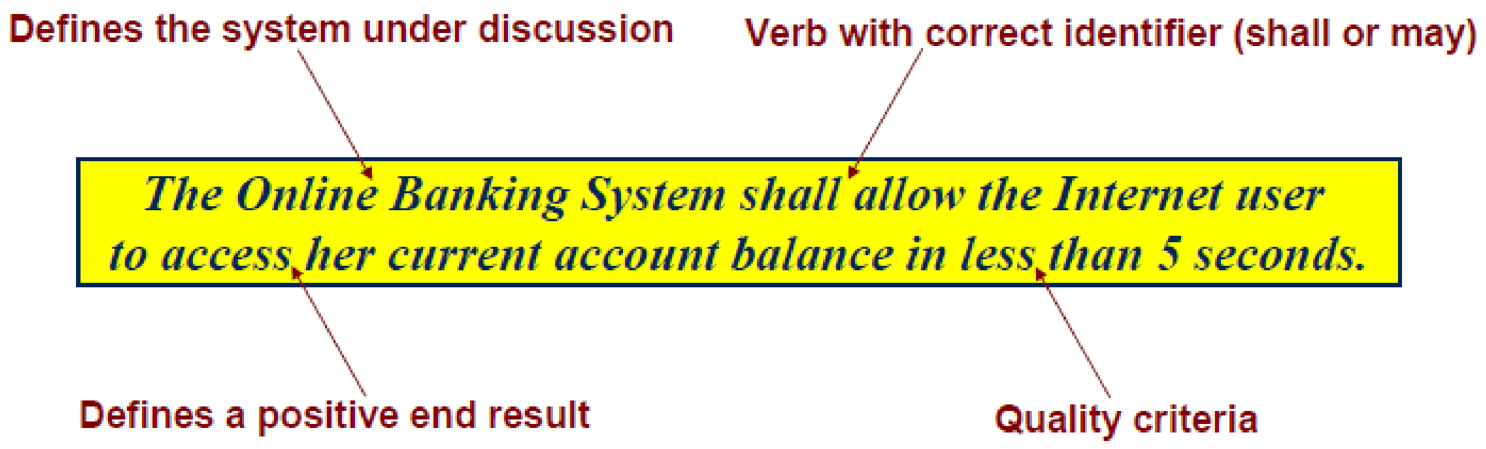
| Property | Measure |
|---|---|
| Speed | Processed transactions/second<br>User/event response time<br>Screen refresh time |
| Size | Mbytes<br>Number of ROM chips |
| Ease of use | Training time<br>Number of help frames |
| Reliability | Mean time to failure<br>Probability of unavailability<br>Rate 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 |

## How to write GOOD REQUIREMENTS

## Constraints of a Good requirement

- Unambiguous
- Testable (verifiable)
- Clear (concise, terse, simple, precise)
- Correct
- Understandable
- Feasible (realistic, possible)
- Independent
- Atomic
- Necessary
- Implementation-free (abstract)

## Writing a good requirement


Defines the system under discussion
Verb with correct identifier (shall or may)
Defines a positive end result
Quality criteria

***The Online Banking System shall allow the Internet user to access her current account balance in less than 5 seconds.***

- Identifies the system under discussion and a desired end result that is wanted within a specified time that is measurable
- The challenge is to seek out the system under discussion, end result, and success measure in every requirement

## Unambiguous

- There should be only one way to interpret the requirement. Sometimes ambiguity is introduced by undefined acronyms:
- ***REQ1 The system shall be implemented using ASP.***
- Does ASP mean Active Server Pages or Application Service Provider? To fix this, we can mention a full name and provide an acronym in parentheses:
- ***REQ1 The system shall be implemented using Active Server Pages (ASP).***
- Here's another example:
- ***REQ1 The system shall not accept passwords longer than 15 characters.***
- It is not clear what the system is supposed to do:
- **The system shall not let the user enter more than 15 characters.**
- **The system shall truncate the entered string to 15 characters.**
- **The system shall display an error message if the user enters more than 15 characters.**

## Testable (Verifiable)

  - *The system shall resist concurrent usage by many users.*
- What number should be considered "many"—10, 100, 1,000?
- Some words can make a requirement untestable
  - Some adjectives: robust, safe, accurate, effective, efficient, expandable, flexible, maintainable, reliable, user-friendly, adequate
  - Some adverbs and adverbial phrases: quickly, safely, in a timely manner
  - Nonspecific words or acronyms: etc., and/or, TBD

## Clear (Concise, Terse, Simple, Precise)

- Requirements should not contain unnecessary verbiage or information. They should be stated clearly and simply:
- *REQ1 Sometimes the user will enter Airport Code, which the system will understand, but sometimes the closest city may replace it, so the user does not need to know what the airport code is, and it will still be understood by the system.*
- This sentence may be replaced by a simpler one:
- *REQ1 The system shall identify the airport based on either an Airport Code or a City Name.*

## Correct

- If a requirement contains facts, these facts should be true:
- *REQ1 Car rental prices shall show all applicable taxes (including 6% state tax).*
- The tax depends on the state, so the provided 6% figure is incorrect.

## Understandable

- Requirements should be grammatically correct and written in a consistent style. Standard conventions should be used. The word “shall” should be used instead of “will,” “must,” or “may.”

## Feasible (Realistic, Possible)

- The requirement should be doable within existing constraints such as time, money, and available resources:
- *REQ1 The system shall have a natural language interface that will understand commands given in English language.*
- This requirement may be not feasible within a short span of development time.

## Independent

- To understand the requirement, there should not be a need to know any other requirement:
- *REQ1 The list of available flights shall include flight numbers, departure time, and arrival time for every leg of a flight.*
- *REQ2 It should be sorted by price.*
- The word “It” in the second sentence refers to the previous requirement. However, if the order of the requirements changes, this requirement will not be understandable.

## Atomic

- The requirement should contain a single traceable element:
- *REQ1 The system shall provide the opportunity to book the flight, purchase a ticket, reserve a hotel room, reserve a car, and provide information about attractions.*
- This requirement combines five atomic requirements, which makes traceability very difficult. Sentences including the words “and” or “but” should be reviewed to see if they can be broken into atomic requirements.

## Necessary

- A requirement is unnecessary if None of the stakeholders needs the requirement. Or Removing the requirement will not affect the system.
- An example of a requirement that is not needed by a stakeholder is a requirement that is added by developers and designers
- For example, the fact that a developer thinks that users would like a feature that displays a map of the airport and he knows how to implement it is not a valid reason to add this requirement.
- An example of a requirement that can be removed because it does not provide any new information might look like the following:
- *REQ1 All requirements specified in the Vision document shall be implemented and tested.*

## Implementation-free (Abstract)

- Requirements should not contain unnecessary design and implementation information:
- *REQ1 Content information shall be stored in a text file.*
- How the information is stored is transparent to the user and should be the designer’s or architect’s decision.

## Consistent

- There should not be any conflicts between the requirements. Conflicts may be direct or indirect. Direct conflicts occur when, in the same situation, different behavior is expected:
- *REQ1 Dates shall be displayed in the mm/dd/yyyy format.*
- *REQ2 Dates shall be displayed in the dd/mm/yyyy format.*
- *REQ1 For users in the U.S., dates shall be displayed in the mm/dd/yyyy format.*
- *REQ2 For users in France, dates shall be displayed in the dd/mm/yyyy format.*

## Nonredundant

- Each requirement should be expressed only once and should not overlap with another requirement:
- *REQ1 A calendar shall be available to help with entering the flight date.*
- *REQ2 The system shall display a pop-up calendar when entering any date.*
- The first requirement (related to only the flight date) is a subset of the second one (related to any date entered by the user).

## Complete

- A requirement should be specified for all conditions that can occur:
- REQ1 A destination country does not need to be displayed for flights within the U.S.
- REQ2 For overseas flights, the system shall display a destination country.
- What about flights to Canada and Mexico? They are neither “within the U.S.” nor“overseas.”
- All applicable requirements should be specified. This is the toughest condition to be checked. There is really no way to be sure that all the requirements are captured and that one week before the production date one of the stakeholders won’t say, “I forgot to mention that I need one more feature in the application.”

## Conflicts

- Conflicts between different nonfunctional requirements are common in
- complex systems, Spacecraft system
- – To minimize weight, the number of separate chips in the system should be minimized
- – To minimize 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 Document

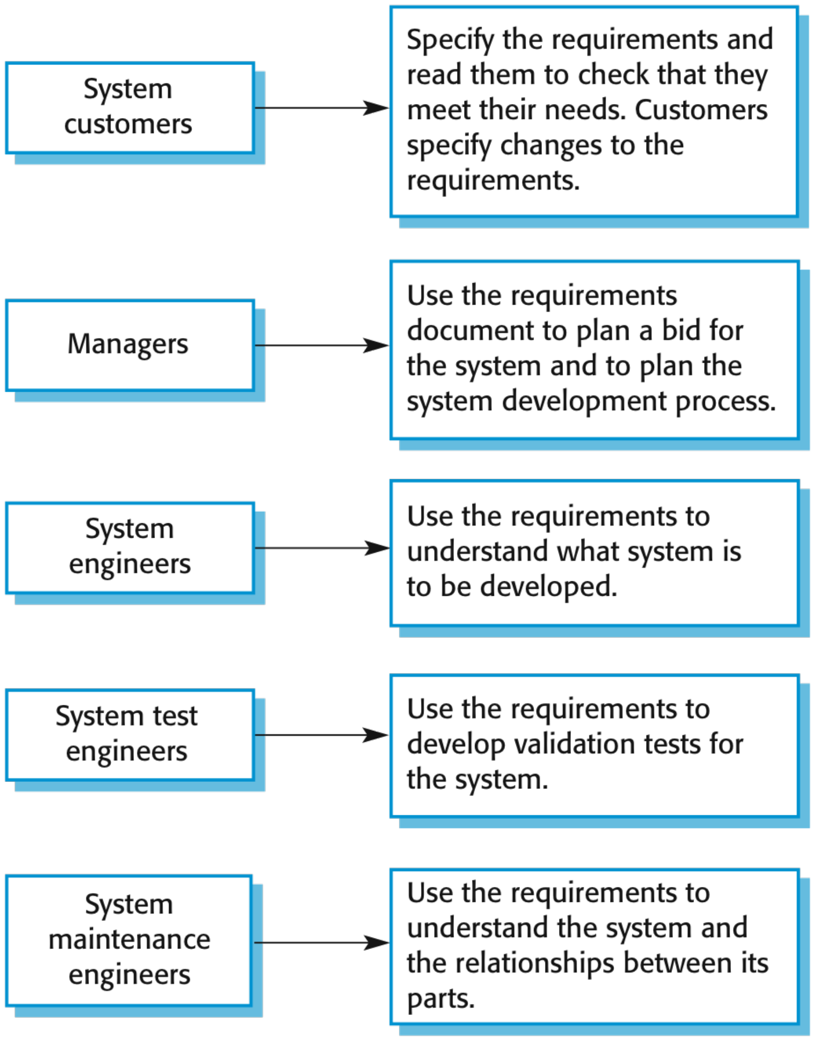
## The software requirements document

- ✧The *software requirements document* is the official statement of what is required of the system developers
- ✧Can include both a definition of user requirements and a specification of the system requirements
- ✧It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it

## Agile methods and requirements

- ✧Many agile methods argue that producing a requirements document is a waste of time as requirements change so quickly
- ✧The document is therefore always out of date
- ✧Methods such as XP use incremental requirements engineering and express requirements as ‘user stories’ (discussed in Chapter 3)
- ✧This is practical for business systems but problematic for systems that require a lot of pre-delivery analysis (e.g. critical systems) or systems developed by several teams

## Users of a requirements document


System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System engineers
Use the requirements to understand what system is to be developed.
System test engineers
Use the requirements to develop validation tests for the system.
System maintenance engineers
Use the requirements to understand the system and the relationships between its parts.

# The structure of a requirements document

| Chapter | Description |
|---|---|
| Preface | This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version. |
| Introduction | This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software. |
| Glossary | This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader. |
| User requirements definition | Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified. |
| System architecture | This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted. |

# The structure of a requirements document

| Chapter | Description |
|---|---|
| System requirements specification | This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined. |
| System models | This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models. |
| System evolution | This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system. |
| Appendices | These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data. |
| Index | Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on. |

## Requirements specification

- ✧The process of writing down the user and system requirements in a requirements document
- ✧User requirements have to be understandable by end-users and customers who do not have a technical background
- ✧System requirements are more detailed requirements and may include more technical information
- ✧The requirements may be part of a contract for the system development
  - It is therefore important that these are as complete as possible

## Ways of writing a system requirements specification

| Notation | Description |
|---|---|
| Natural language | The requirements are written using numbered sentences in natural language. Each sentence should express one requirement. |
| Structured natural language | The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement. |
| Design description languages | This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications. |
| Graphical notations | Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used. |
| Mathematical specifications | These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract |

## Requirements and design

- ✧In principle, requirements should state what the system should do and the design should describe how it does this.
- ✧In practice, requirements and design are inseparable
  - A system architecture may be designed to structure the requirements;
  - The system may inter-operate with other systems that generate design requirements;
  - The use of a specific architecture to satisfy non-functional requirements may be a domain requirement.
  - This may be the consequence of a regulatory requirement.

## Natural language specification

- ✧Requirements are written as natural language sentences supplemented by diagrams and tables
- ✧Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.

## Guidelines for writing requirements

- ✧Create a standard format and use it for all requirements
- ✧Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.
- ✧Use text highlighting to identify key parts of the requirement
- ✧Avoid the use of computer jargon
- ✧Include an explanation (rationale) of why a requirement is necessary

## Problems with natural language

- ✧Lack of clarity
  - Precision is difficult without making the document difficult to read.
- ✧Requirements confusion
  - Functional and non-functional requirements tend to be mixed-up.
- ✧Requirements amalgamation
  - Several different requirements may be expressed together.

## Example requirements for the insulin pump software system

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)*

## Structured specifications

- An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way
- This works well for some types of requirements, e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements

## 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
- ✧Information about the information needed for the computation and other entities used
- ✧Description of the action to be taken
- ✧Pre and post conditions (if appropriate)
- ✧The side effects (if any) of the function

## A structured specification of a requirement for an insulin pump

*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—the dose in insulin to be delivered.

**Destination** Main control loop.

## A structured specification of a requirement for an insulin pump

**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.

**Requirements**

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
- ✧For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios

## Tabular specification of computation for an insulin pump

| 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 |

## Requirements engineering processes

- ✧The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements
- ✧However, there are a number of generic activities common to all processes
  - Requirements elicitation
  - Requirements analysis
  - Requirements validation
  - Requirements management
- ✧In practice, RE is an iterative activity in which these processes are interleaved

## Requirements elicitation and analysis

- ✧Sometimes called *requirements elicitation* or *requirements discovery*
- ✧Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints
- ✧May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*

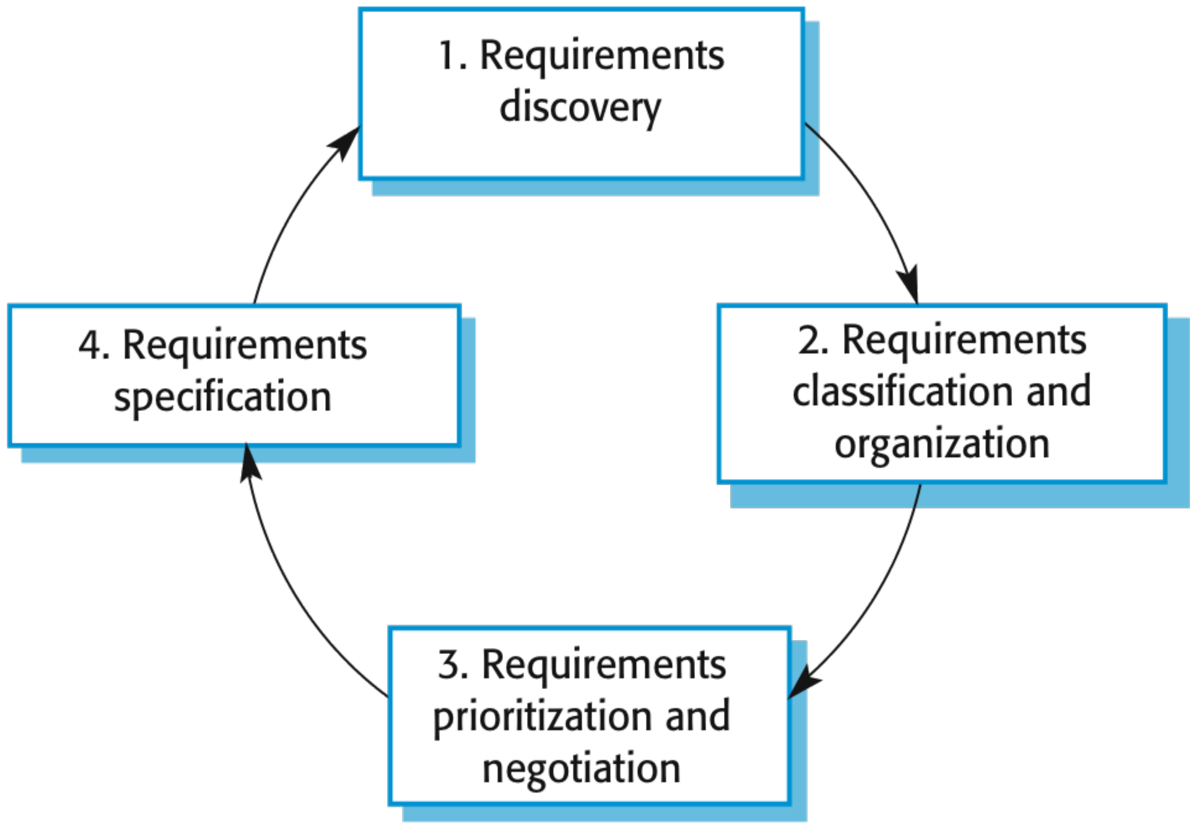
## Problems of requirements analysis

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

## Requirements elicitation and analysis

- ✧Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.
- ✧Stages include:
  - Requirements discovery
  - Requirements classification and organization
  - Requirements prioritization and negotiation
  - Requirements specification

## The requirements elicitation and analysis process


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

## Process activities

- ✧Requirements discovery
  - Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage
- ✧Requirements classification and organization
  - Groups related requirements and organizes them into coherent clusters
- ✧Prioritization and negotiation
  - Prioritizing requirements and resolving requirements conflicts
- ✧Requirements specification
  - Requirements are documented and input into the next round of the spiral

## REQUIREMENTS ELICITATION

## BABOK Guidelines

- Brainstorming
- Document Analysis
- Focus Groups
- Interface Analysis
- Interviews
- Observation
- Prototyping
- Requirements Workshops
- Survey/Questionnaire

## Requirements Elicitation techs

***Traditional techniques*** questionnaires and surveys, interviews, and analysis of existing documentation such as organisational charts, process models or standards, and user or other manuals of existing systems.

***Group elicitation techniques*** They include brainstorming and focus groups, as well as RAD/JAD workshops (using consensus-building workshops with an unbiased facilitator)

***Contextual techniques*** ethnographic techniques such as participant observation.

## Requirements Elicitation techs…

***Cognitive techniques*** *protocol analysis* (in
which an expert thinks aloud while performing a task,
to
provide the observer with insights into the cognitive
processes
used to perform the task),
***laddering*** (using probes to elicit structure and
content of stakeholder knowledge),
***card sorting*** (asking stakeholders to sort cards in
groups, each of which has name of some domain
entity),
***repertory grids*** (constructing an attribute matrix for
entities, by asking stakeholders for attributes
applicable to entities and values for cells in each

## *Laddering*

- It begins with a simple question, and then another question is asked about that response.
- For example, an interviewer may ask:
  - "How come you skipped class?" and the response may be: "I went out with my friends".
  - The next question would be something like "Why did you go out with your friends?“

This technique can very be tiring and or boring for the interviewee

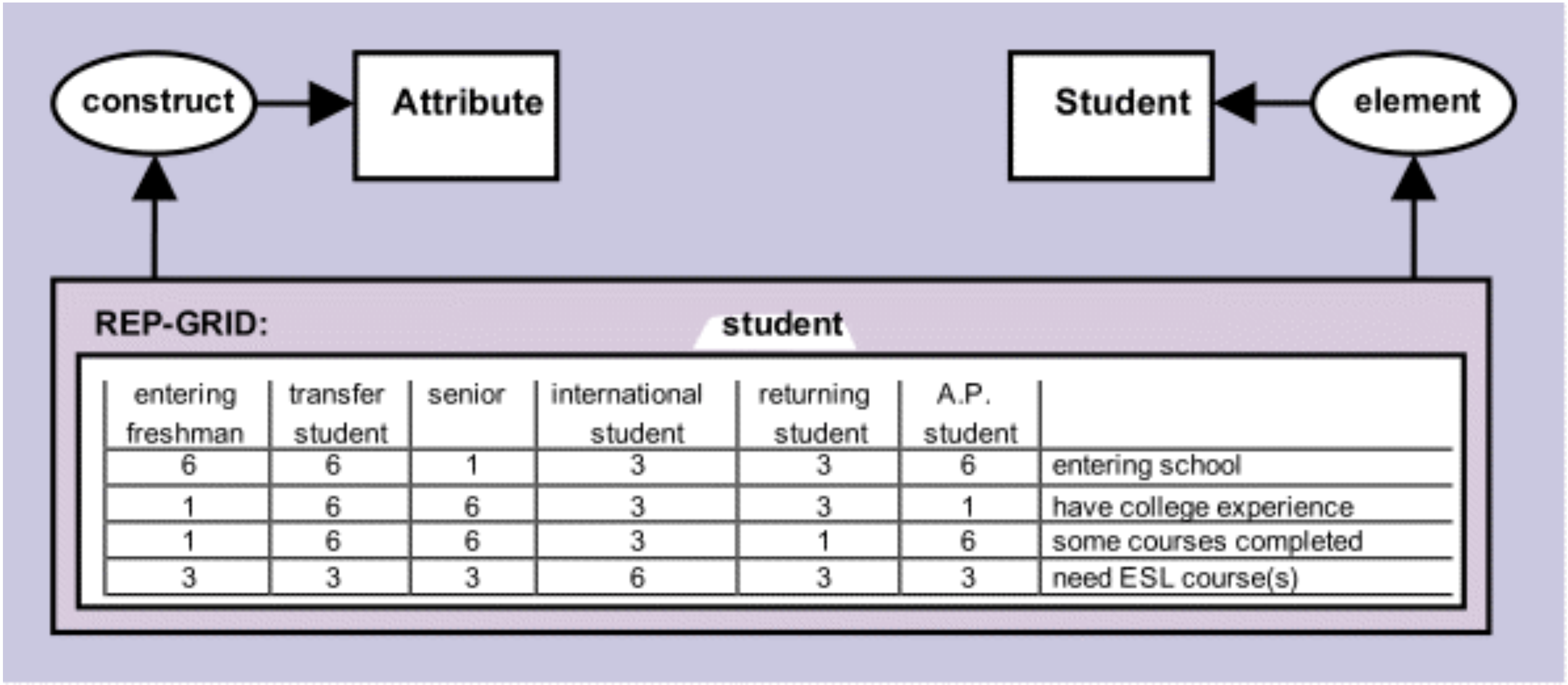
## Laddering example

Interviewer: "Why x?"
Subject: "Because z"
Interviewer: "Why z?"
Subject: "Because b"
Interviewer: "Why b?"
The first responses are generally functional justifications,
like "I went out with my friends because I wanted some pizza",
or "I wanted some pizza because I used to eat it as a child"; but eventually the interviewer hopes to reach a virtue justification like
"It's good to be childish". Then it is fair to conclude that the interviewee skipped class because he valued childishness.

## Repertory Grid

- The main components of the repertory grid are:
- **The topic** – what the interview is about.
- **Elements** – these are examples that illustrate the topic. They can be people, objects, experiences, events, according to the topic. The elements can either be chosen by the interviewee, or they can be preselected. In the example about students' views of lecturers given above, the topic would be "What makes a good lecturer?" and the elements would be particular lecturers known to the interviewee. If a number of students were being interviewed, with different backgrounds, the lecturers/elements would not necessarily be the same.
- **Constructs** – the most important component of the repertory grid. This is where the elements are compared with one another to produce a series of statements which describe what the interviewee thinks about the topic. These statements will form the eventual unit of analysis. They will be bipolar – in other words, every statement will be presented as opposite ends of a pole. The students being interviewed about lecturers might say that Lecturer A, as opposed to Lecturers B and C, explains things clearly whereas B and C are hard to follow. So, one set of constructs would be "explains things clearly" as opposed to "hard to follow".
- **Ratings** – once the main constructs and elements are in place, they are entered on a grid with the elements on top and the constructs down the side. The interviewee then rates each element against each construct according to a rating scale, usually of 1-5

# Repertory Grid



| entering freshman | transfer student | senior | international student | returning student | A.P. student | |
|---|---|---|---|---|---|---|
| 6 | 6 | 1 | 3 | 3 | 6 | entering school |
| 1 | 6 | 6 | 3 | 3 | 1 | have college experience |
| 1 | 6 | 6 | 3 | 1 | 6 | some courses completed |
| 3 | 3 | 3 | 6 | 3 | 3 | need ESL course(s) |

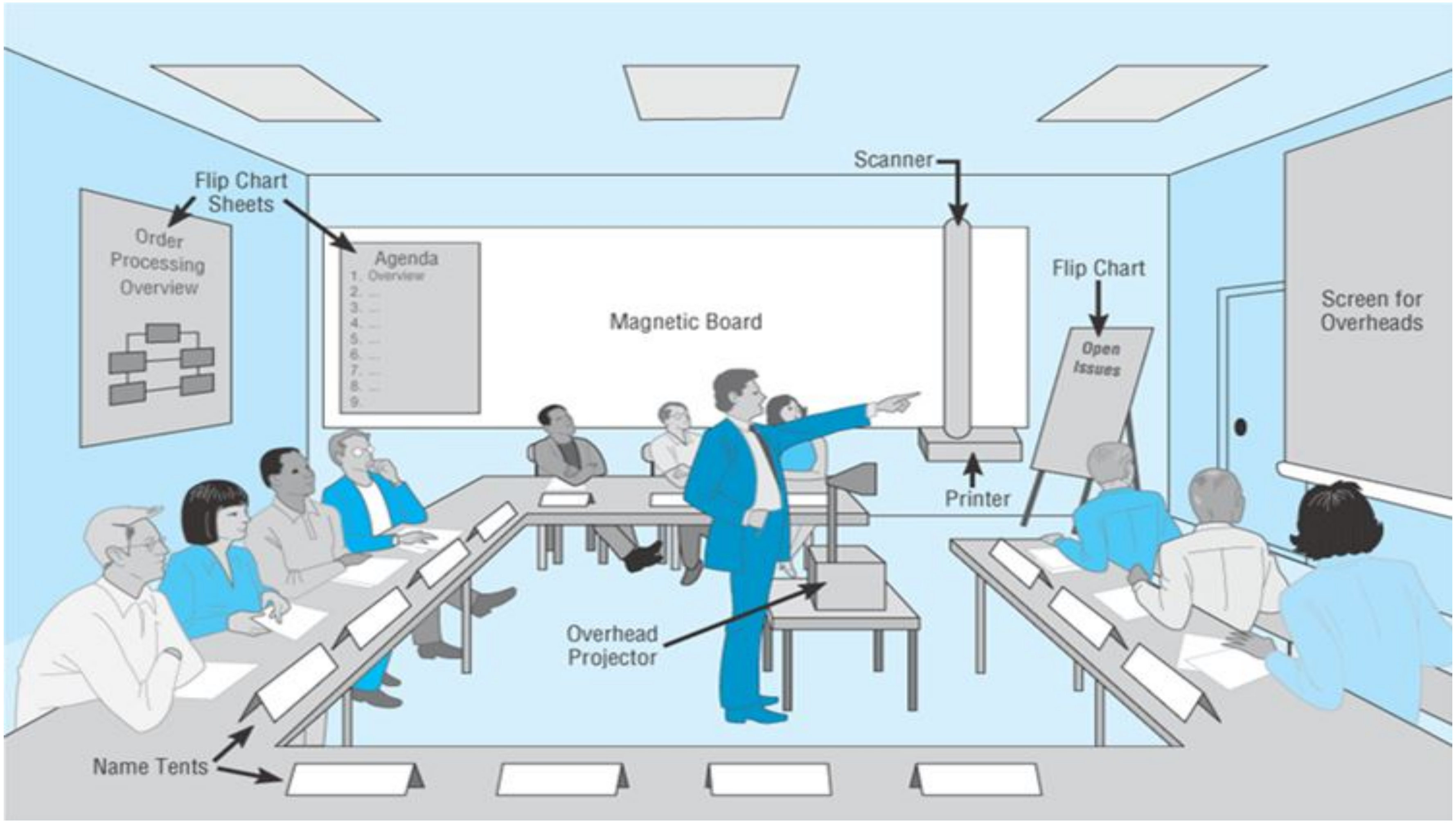
## Uses of repertory grid

- Because of its ability to capture good data, the repertory grid is used in a wide range of contexts. Below is a non-exhaustive list:
- human resources (for example performance appraisals, job analysis, training needs analysis, staff and organizational development),
- psychology (for example, psychological tests or counselling type interviews),
- brand analysis and consumer behaviour,
- team development and organizational studies,
- information retrieval studies and systems analysis, for example mental modelling.

## Advantages and drawbacks of repertory grid

- Despite the flexibility of the technique it does have drawbacks:
- It is time-consuming; each interview will take up to an hour.
- It can appear rather artificial, and senior managers in particular may be sceptical about its value, and hence rather unwilling to give time to it.
- There are many variations of design and it can be difficult to select the right one.
- The analysis process can "overwhelm with numbers" and one can become fascinated by the process of computer analysis and what it can "discover" – at the expense of the "bigger picture".

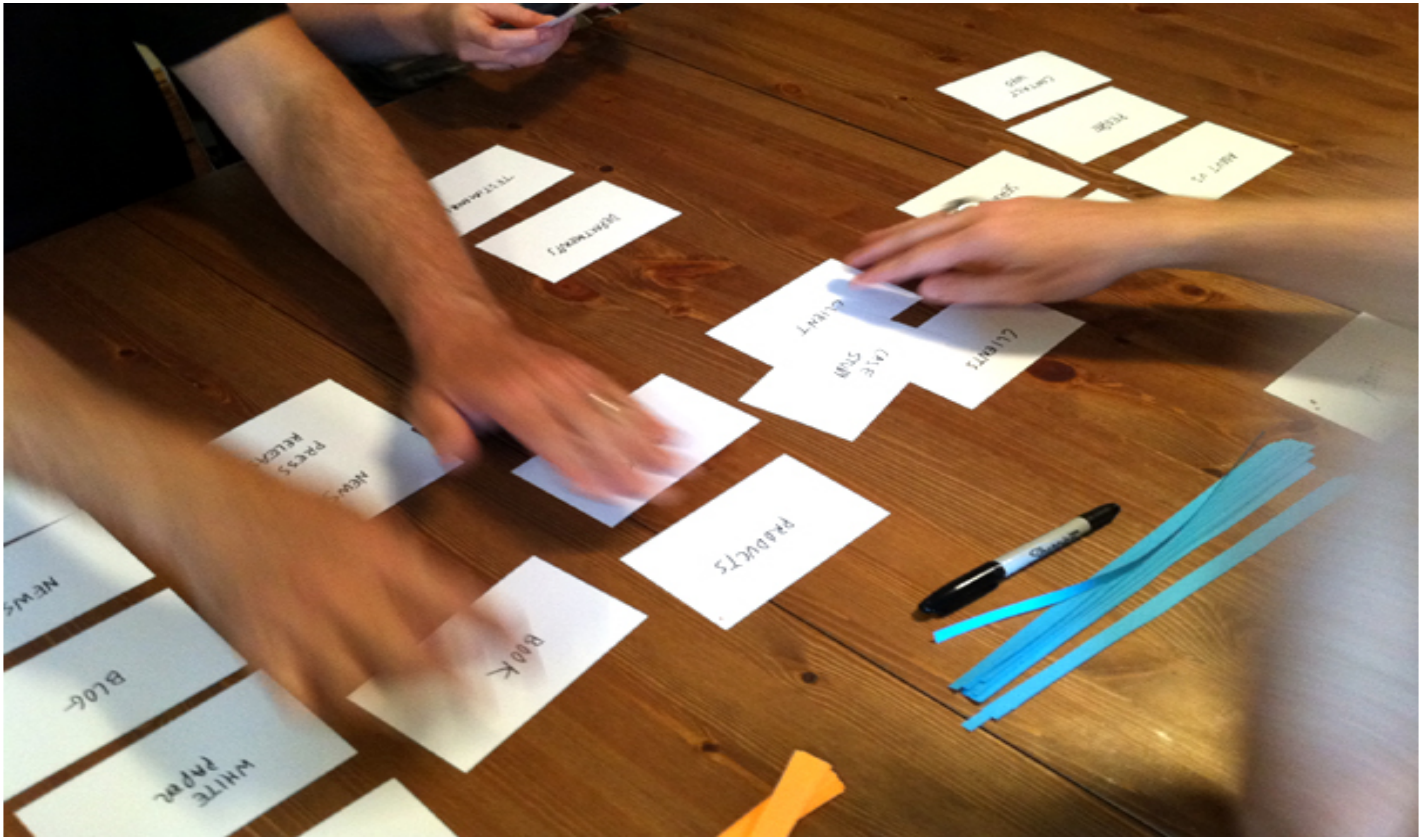
# JAD Sessions



## Focused Groups

BRAND CAMP
by Tom Fishburne
OH ALL-KNOWING FOCUS GROUP, TELL ME YOUR UNMET CONSUMER NEEDS, SO THAT WE CAN CREATE DISRUPTIVE INNOVATIONS FOR YOU.
MORE PROMOTIONS
YEAH, MAKE EVERYTHING CHEAPER
©2010
TOMFISHBURNE.COM

# Card Sorting



## Ethnography

- ✧A social scientist spends a considerable time observing and analysing how people actually work
- ✧People do not have to explain or articulate their work
- ✧Social and organizational factors of importance may be observed
- ✧Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models

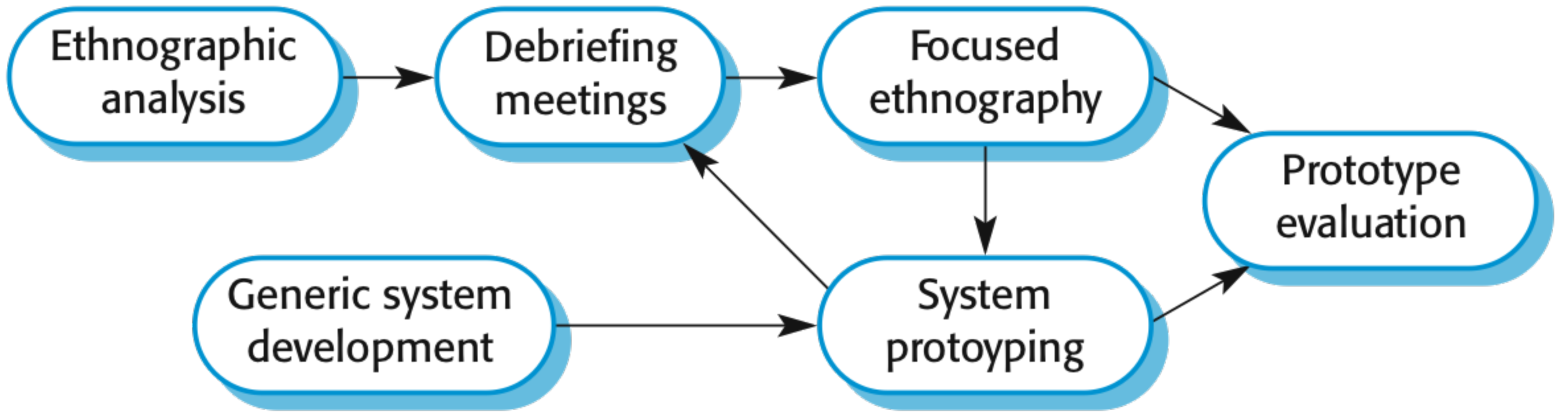
## Scope of ethnography

- ✧Requirements that are derived from the way that people actually work rather than the way I which process definitions suggest that they ought to work
- ✧Requirements that are derived from cooperation and awareness of other people's activities
  - Awareness of what other people are doing leads to changes in the ways in which we do things
- ✧Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system

## Focused ethnography

- ✧Developed in a project studying the air traffic control process
- ✧Combines ethnography with prototyping
- ✧Prototype development results in unanswered questions which focus the ethnographic analysis
- ✧The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant

## Ethnography and prototyping for requirements analysis



## Requirements validation

- ✧Concerned with demonstrating that the requirements define the system that the customer really wants
- ✧Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error

## 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
- ✧Verifiability. Can the requirements be checked?

## Requirements validation techniques

- ✧Requirements reviews
  - Systematic manual analysis of the requirements
- ✧Prototyping
  - Using an executable model of the system to check requirements. Covered in Chapter 2
- ✧Test-case generation
  - Developing tests for requirements to check testability

## 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?

## Requirements management

- ✧*Requirements management* is the process of managing changing requirements during the requirements engineering process and system development
- ✧New requirements emerge as a system is being developed and after it has gone into use
- ✧You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.

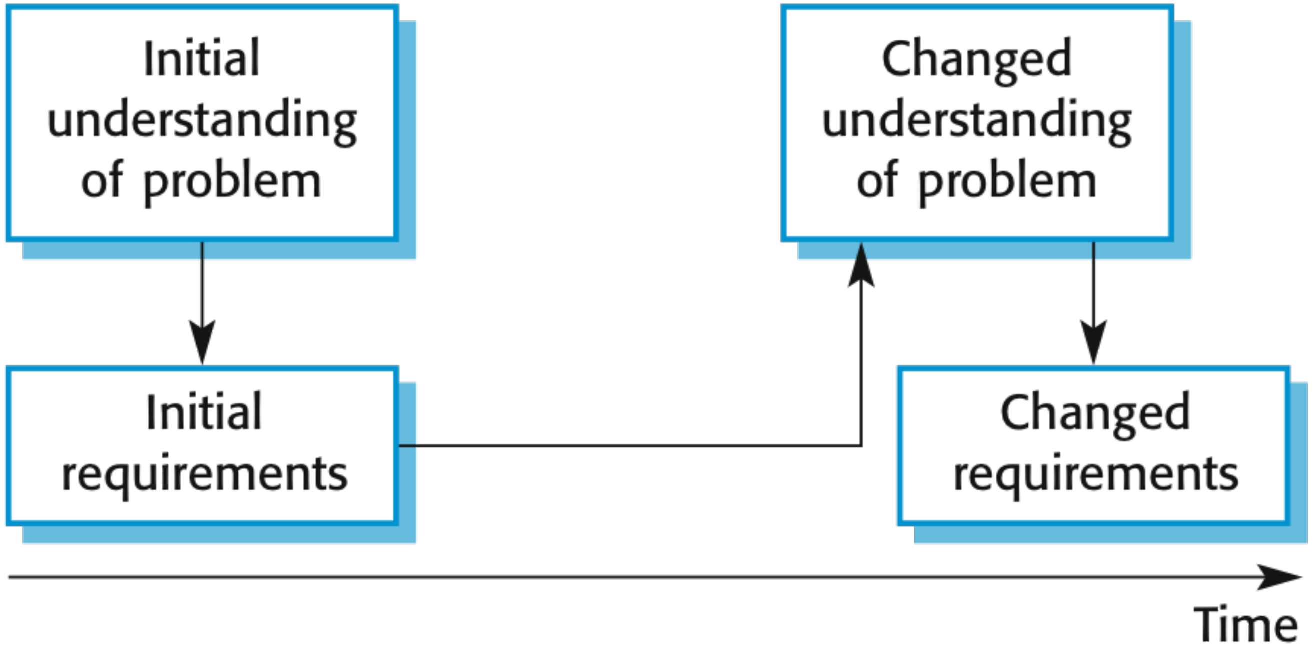
## Changing requirements

- ✧The business and technical environment of the system always changes after installation
  - ▪ New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by
- ✧The people who pay for a system and the users of that system are rarely the same people
  - ▪ System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals

## Changing requirements

- Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory
  - The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed

## Requirements evolution


Initial understanding of problem
Changed understanding of problem
Initial requirements
Changed requirements
Time

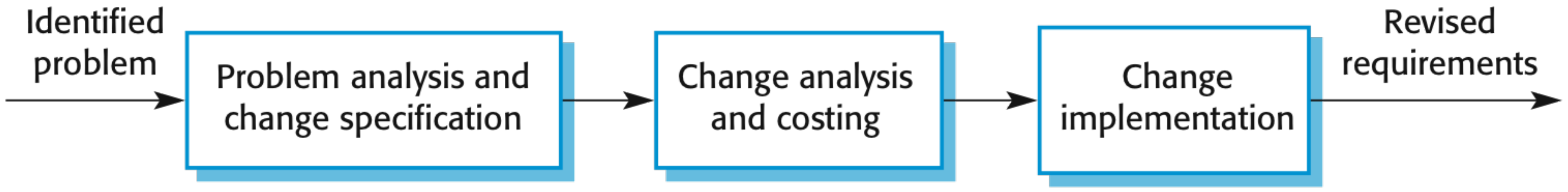
## Requirements management planning

- ✧Establishes the level of requirements management detail that is required.
- ✧Requirements management decisions:
  - Requirements identification Each requirement must be uniquely identified so that it can be cross-referenced with other requirements
  - A change management process This is the set of activities that assess the impact and cost of changes. I discuss this process in more detail in the following section.
  - Traceability policies These policies define the relationships between each requirement and between the requirements and the system design that should be recorded
  - Tool support Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems

## Requirements change management

- Deciding if a requirements change should be accepted
  - Problem analysis and change specification
    - During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
  - Change analysis and costing
    - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
  - Change implementation
    - The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.

## Requirements change management


Identified problem
Problem analysis and change specification
Change analysis and costing
Change implementation
Revised requirements

## Key points

- ✧Requirements for a software system set out what the system should do and define constraints on its operation and implementation
- ✧Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out
- ✧Non-functional requirements often constrain the system being developed and the development process being used
  - ✧They often relate to the emergent properties of the system and therefore apply to the system as a whole

## Key points

- ✧The software requirements document is an agreed statement of the system requirements. It should be organized so that both system customers and software developers can use it.
- ✧The requirements engineering process is an iterative process including requirements elicitation, specification and validation
- ✧Requirements elicitation and analysis is an iterative process that can be represented as a spiral of activities – requirements discovery, requirements classification and organization, requirements negotiation and requirements documentation

## Key points

- ✧You can use a range of techniques for requirements elicitation including interviews, use-cases and ethnography
- ✧Requirements validation is the process of checking the requirements for validity, consistency, completeness, realism and verifiability
- ✧Business, organizational and technical changes inevitably lead to changes to the requirements for a software system. Requirements management is the process of managing and controlling these changes.

## Trello