

Elicitation techniques

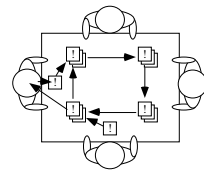
- **Asking:**
 - interview
 - Delphi technique
 - brainstorming session
- **Observing**
 - task analysis
 - scenario analysis
 - ethnography
 - form analysis
 - synthesis from existing system
- **Others:**
 - analysis of natural language descriptions
 - domain analysis
 - Business Process Redesign (BPR)
 - prototyping

Interviewing

- **Conduct a series of interviews**
 - Ask about specific details
 - Ask about the stakeholder's vision for the future
 - Ask if they have alternative ideas
 - Ask for other sources of information
 - Ask them to draw diagrams

Brainstorming

- Appoint an experienced moderator
- Arrange the attendees around a table
- Decide on a 'trigger question'
- Ask each participant to write an answer and pass the paper to its neighbour



- *Joint Application Development (JAD)* is a technique based on intensive brainstorming sessions

Observation

- Read documents and discuss requirements with users
- Shadowing important potential users as they do their work
 - ask the user to explain everything he or she is doing
- Session video taping

Task Analysis

- Task analysis is the process of analyzing the way people perform their jobs: the things they do, the things they act on and the things they need to know.
- The relation between tasks and goals: a task is performed in order to achieve a goal.
- Task analysis has a broad scope.

Task Analysis

- Task analysis concentrates on the current situation. However, it can be used as a starting point for a new system:
 - users will refer to new elements of a system and its functionality
 - scenario-based analysis can be used to exploit new possibilities

Scenario-Based Analysis

- Provides a more user-oriented view perspective on the design and development of an interactive system.
- The defining property of a scenario is that it projects a concrete description of an activity that the user engages in when performing a specific task, a description sufficiently detailed so that the design implications can be inferred and reasoned about.

Scenario-Based Analysis (example)

- first shot:
 - check due back date
 - if overdue, collect fine
 - record book as being available again
 - put book back
- as a result of discussion with library employee:
 - what if person returning the book is not registered as a client?
 - what if the book is damaged?
 - how to handle in case the client has other books that are overdue, and/or an outstanding reservation?

Scenario-Based Analysis

Scenario view

- concrete descriptions
- focus on particular instances
- work-driven
- open-ended, fragmentary
- informal, rough, colloquial
- envisioned outcomes

Standard view

- abstract descriptions
- focus on generic types
- technology-driven
- complete, exhaustive
- formal, rigorous
- specified outcomes

Form analysis

Proceedings request form:

Client name
Title
Editor
Place
Publisher
Year

Certainty vs uncertainty

Prototyping

- The simplest kind: *paper prototype*.
 - a set of pictures of the system that are shown to users in sequence to explain what would happen
- The most common: a mock-up of the system's UI
 - Written in a rapid prototyping language
 - Does *not* normally perform any computations, access any databases or interact with any other systems
 - May prototype a particular aspect of the system

Use case analysis

- Determine the classes of users that will use the facilities of this system (actors)
- Determine the tasks that each actor will need to do with the system

Requirement documents

- An informal outline of the requirements using a few paragraphs or simple diagrams
 - requirements *definition*
- A long list of specifications that contain thousands of pages of intricate detail
 - requirements *specification*
- Requirements documents for large systems are normally arranged in a hierarchy

Requirement documents

- Level of required detail
 - The size of the system
 - The need to interface to other systems
 - The readership
 - The stage in requirements gathering
 - The level of experience with the domain and the technology
 - The cost that would be incurred if the requirements were faulty

Prioritizing requirements (MoSCoW)

- **Must haves:** top priority requirements
- **Should haves:** highly desirable
- **Could haves:** if time allows
- **Won't haves:** not today

One dimensional

Prioritizing requirements (Kano model)

- **Attractive:** more satisfied if +, not less satisfied if –
- **Must-be:** dissatisfied when -, at most neutral
- **One-dimensional:** satisfaction proportional to number
- **Indifferent:** don't care
- **Reverse:** opposite of what analyst thought
- **Questionable:** preferences not clear

Kano model



Requirements specification

- readable
- understandable
- non-ambiguous
- complete
- verifiable
- consistent
- modifiable
- traceable
- usable
- ...

IEEE Standard 830

1. Introduction

- 1.1. Purpose
- 1.2. Scope
- 1.3. Definitions, acronyms and abbreviations
- 1.4. References
- 1.5. Overview

2. General description

- 2.1. Product perspective
- 2.2. Product functions
- 2.3. User characteristics
- 2.4. Constraints
- 2.5. Assumptions and dependencies

3. Specific requirements

IEEE Standard 830 (cntd)

3. Specific requirements

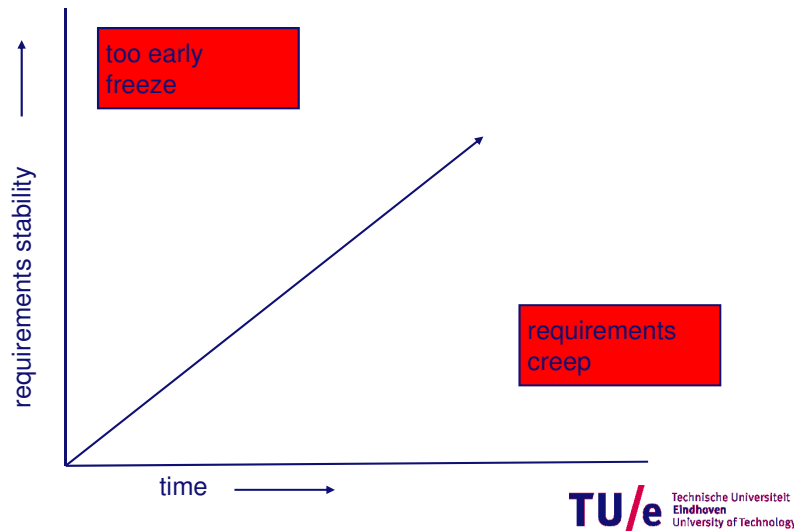
- 3.1. External interface requirements
 - 3.1.1. User interfaces
 - 3.1.2. Hardware interfaces
 - 3.1.3. Software interfaces
 - 3.1.4. Comm. interfaces

3.2. Functional requirements

- 3.2.1. User class 1
 - 3.2.1.1. Functional req. 1.1
 - 3.2.1.2. Functional req. 1.2
- ...
- 3.2.2. User class 2
- ...

- 3.3. Performance requirements
- 3.4. Design constraints
- 3.5. Software system attributes
- 3.6. Other requirements

Requirements management



SE, Requirements Engineering, Hans van Vliet, ©2007

20

Requirements management

- Requirements identification (number, goal-hierarchy numbering, version information, attributes)
- Requirements change management (CM)
- Requirements traceability:
 - Where is requirement implemented?
 - Do we need this requirement?
 - Are all requirements linked to solution elements?
 - What is the impact of this requirement?
 - Which requirement does this test case cover?
- Related to Design Space Analysis

TU/e Technische Universiteit Eindhoven University of Technology

SE, Requirements Engineering, Hans van Vliet, ©2007

21

The 7 sins of the analyst

- noise
- silence
- overspecification
- contradictions
- ambiguity
- forward references
- wishful thinking

TU/e Technische Universiteit Eindhoven University of Technology

SE, Requirements Engineering, Hans van Vliet, ©2007

22

Functional vs. Non-Functional Requirements

- functional requirements: the system services which are expected by the users of the system.
- non-functional (quality) requirements: the set of constraints the system must satisfy and the standards which must be met by the delivered system.
 - speed
 - size
 - ease-of-use
 - reliability
 - robustness
 - portability

TU/e Technische Universiteit Eindhoven University of Technology

SE, Requirements Engineering, Hans van Vliet, ©2007

23

Reviewing requirements

- Each individual requirement should
 - Have benefits that outweigh the costs of development
 - Be important for the solution of the current problem
 - Be expressed using a clear and consistent notation
 - Be unambiguous
 - Be logically consistent
 - Lead to a system of sufficient quality
 - Be realistic with available resources
 - Be verifiable
 - Be uniquely identifiable
 - Does not over-constrain the design of the system

Validation of requirements

- Inspection of the requirement specification w.r.t. correctness, completeness, consistency, accuracy, readability, and testability.
- Some aids:
 - structured walkthroughs
 - prototypes
 - simulation
 - use cases and scenarios analysis
 - develop a test plan
 - tool support for formal specifications

Requirements Review Checklist

1. Does the (software) product have a succinct name, and a clearly described purpose?
2. Are the characteristics of users and of typical usage mentioned? (No user categories missing.)
3. Are all external interfaces of the software explicitly mentioned? (No interfaces missing.)
4. Does each specific requirement have a unique identifier?
5. Is each requirement atomic and simply formulated? (Typically a single sentence. Composite requirements must be split.)
6. Are requirements organized into coherent groups? (If necessary, hierarchical; not more than about ten per group.)
7. Is each requirement prioritized? (Is the meaning of the priority levels clear?)

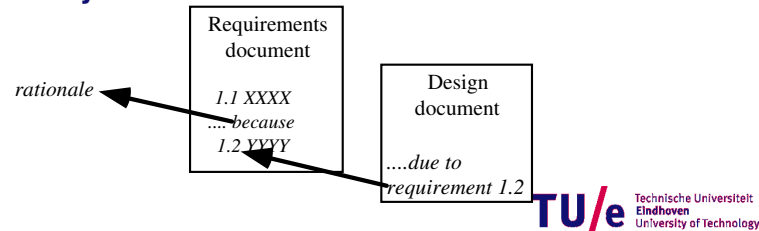
Requirements Review Checklist (continued)

8. Are all unstable requirements marked as such? (TBC= 'To Be Confirmed', TBD= 'To Be Defined')
9. Is each requirement verifiable (in a provisional acceptance test)? (Measurable: where possible, quantify; capacity, performance, accuracy)
10. Are the requirements consistent? (Non-conflicting.)
11. Are the requirements sufficiently precise and unambiguous? (Which interfaces are involved, who has the initiative, who supplies what data, no passive voice.)
12. 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.)
13. Are the requirements understandable to those who will need to work with them later?
14. Are the requirements realizable within budget?
15. Do the requirements express actual customer needs (in the language of the problem domain), rather than solutions (in developer jargon)?

Requirements documents...

- The document should be:
 - sufficiently complete
 - well organized
 - clear
 - agreed to by all the stakeholders

- Traceability:



Requirements document...

- A. Problem
- B. Background information
- C. Environment and system models
- D. Functional Requirements
- E. Non-functional requirements

Managing Changing Requirements

- Requirements change because:
 - Business process changes
 - Technology changes
 - The problem becomes better understood
- Requirements analysis never stops
 - Continue to interact with the clients and users
 - The benefits of changes must outweigh the costs.
 - Certain small changes (e.g. look and feel of the UI) are usually quick and easy to make at relatively little cost.
 - Larger-scale changes have to be carefully assessed
 - Forcing unexpected changes into a partially built system will probably result in a poor design and late delivery
 - Some changes are enhancements in disguise
 - Avoid making the system *bigger*, only make it *better*