Requirements Engineering

Mark van den Brand
Tom Verhoeff

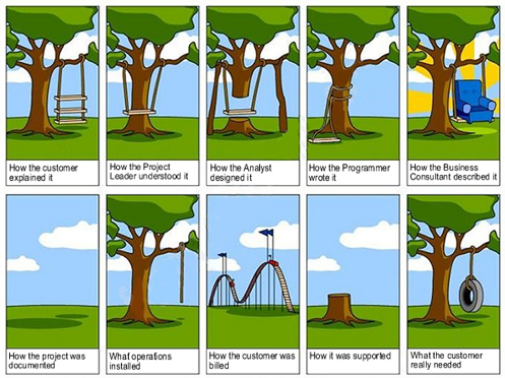
TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Questions

- Why does prototyping fall between waterfall and agile?
- What is the process model of the Software Engineering Projects?
- Does an agile process model deliver maintainable software? Give a couple of (motivated) arguments.

Requirements Engineering



Domain Analysis

- The process by which a software engineer learns about the domain to better understand the problem:
  - The *domain* is the general field of business or technology in which the clients will use the software
  - A *domain expert* is a person who has a deep knowledge of the domain
- Benefits of performing domain analysis:
  - Faster development
  - Better system
  - Anticipation of extensions

Domain analysis

Domain analysis document:

A. **Introduction**
B. **Glossary**
C. **General knowledge about the domain**
D. **Customers and users**
E. **The environment**
F. **Tasks and procedures currently performed**
G. **Competing software**
H. **Similarities to other domains**

Domain analysis

Example document:

http://www.site.uottawa.ca/~laganier/seg3700/cemdomain.htm

Domain analysis

A. Introduction
This document describes background information that has been gathered about events in organizations and how they are handled. This information is to be used to guide the development of software to automate the process of informing peopl

B. Glossary
- **Event:** A meeting, a social occasion or an activity involving a significant number of employees. Several categories of events have been identified:
  - **Open event:** An event that starts at a precise instant but with no predetermined duration. Meetings and celebrations often fall into this category.
  - **Fixed event:** An event that starts at a precise instant and with a predetermined duration. Course lectures and seminars are examples of this kind of event.
  - **Day events:** An event associated with a particular day without precise start and end times. Birthday, thematic journey are such events.
  - **Recurrent event:** An event that occurs repeatedly on some regular schedule (for example daily, weekly or monthly). The event normally has a starting date and an ending date. Courses and social activities are often recurrent eve
  - **Composite event:** An event composed of several sub-events. For example, a training activity can be composed of a registration period (fixed event), a series of seminars (recurrent events), and a final evening celebration (open ev

C. General knowledge about the domain
- Most events occur during working days.
- Events are generally associated with a location (where the event is to be held).
- The name of a contact person is often associated with an event. That person is the one that organize the event or that can give complementary information about the event.
- Group of interest are often created to target more precisely peoples that might be interested by a certain event.
- Outdated events are of little interest.
- Each event has a title, a location and the name of a contact person associated with it.
- Events may be seen by anyone within the organization, but there should be some control over posting events to reduce the risk of duplicate postings and other clutter.

D. Clients and users
Potential clients are medium or large companies whose staff use computers to perform many kinds of daily work. Others impacted by the system will include:
- Employees at all levels have an interest in events and are potential users of event manager software. These employees range from computer novices to sophisticated programmers; however they all have a computer on their desk, have acc to use a web browser. They have been exposed to and accept new technology but are subject to tight time constraints and have little time to learn and customize new software tools.
- A system administrator normally manages the computer environment.
- Technicians typically install software that must be available to all users.

E. The environment
The actors all have a computer on their desks; it is most common for this to be MS-Windows based, but a significant minority of potential clients use other platforms.

A wide variety of software is installed on these computers, with each actor having a unique configuration. Some software may be installed on every computer using a site-wide license.

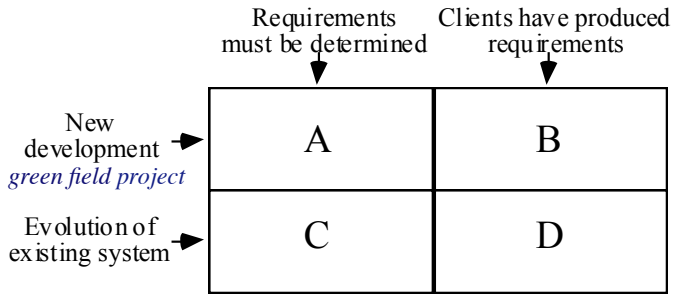
F. Tasks and procedures currently performed
- Informing others about events: The organizer of the event (or someone who has heard about it) compiles information concerning an event and posts it so members of the organization can see it.

One approach is to post the event on a physical bulletin board at a place that most users pass each day. In a large organization, this method is too unreliable due to the sheer volume of events.

Another approach is to send email and paper leaflets to all members of the staff without knowing exactly who will be interested. In some cases there are mailing lists to make event notification more selective, however mailing lists are of might be interested never find out about an event.

In either approach, there is typically no central listing of available events and the information about an event is not presented in a standard fashion.

Starting Point for Software Projects

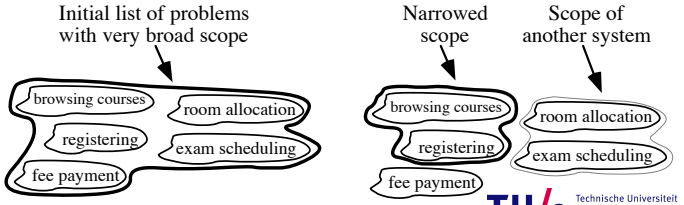


Defining Problem and Scope

- **A problem can be expressed as:**
  - **A *difficulty* the users or customers are facing,**
  - **Or as an *opportunity* that will result in some benefit such as improved productivity or sales.**
- **The solution to the problem normally will entail developing software**
- **A good problem statement is short and succinct**

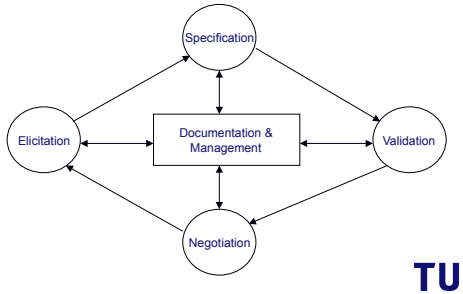
Defining the Scope

- Narrow the *scope* by defining a more precise problem
  - List all the things you might imagine the system doing
    - Exclude some of these things if too broad
    - Determine high-level goals if too narrow
- Example: A university registration system



Processes in requirements engineering

- Requirements elicitation
- Requirements specification
- Requirements validation and verification
- Requirements negotiation



What is a Requirement ?

- It is a statement describing either
  - 1) an aspect of what the proposed system must do,
  - or 2) a constraint on the system's development.
  - In either case it must contribute in some way towards adequately solving the customer's problem;
  - the set of requirements as a whole represents a negotiated agreement among the stakeholders.
- A collection of requirements is a *requirements document*.

Types of Requirements

- Functional requirements
  - Describe *what* the system should do
- Quality requirements
  - C*onstraints* on the design to meet specified levels of quality
- Platform requirements
  - C*onstraints* on the environment and technology of the system
- Process requirements
  - C*onstraints* on the project plan and development methods

Functional Requirements

- What *inputs* the system should accept
- What *outputs* the system should produce
- What data the system should *store* that other systems might use
- What *computations* the system should perform
- The *timing and synchronization* of the above

Quality Requirements

- All must be verifiable
- Examples: Constraints on
  - Response time
  - Throughput
  - Resource usage
  - Reliability
  - Availability
  - Recovery from failure
  - Allowances for maintainability and enhancement
  - Allowances for reusability

Vragen

- Why is *domain analysis* crucial for good requirements?
- Why is scoping important in problem definition?
- What is the difference between functional and non-functional requirements?
- What is the relation between requirements and testing?

Conceptual modeling

- You model part of reality: the Universe of Discourse (UoD)
- This model is an explicit conceptual model
- People in the UoD have an implicit conceptual model of that UoD
- Making this implicit model explicit poses problems:
  - analysis problems
  - negotiation problems

Conceptual modeling

- Requirements engineering is difficult
- Success depends on the degree with which we manage to properly describe the system desired

Conceptual modeling

- Beware of subtle mismatches:
  - a library employee may also be a client
  - there is a difference between `a book` and `a copy of a book`
  - status info `present` / `not present` is not sufficient; a (copy of a) book may be lost, stolen, in repair, ...

Conceptual modeling

- Humans as sources of information:
  - different backgrounds
  - short-term vs long-term memory
  - human prejudices
  - limited capability for rational thinking

Conceptual modeling

- How we study the world around us:
  - people have a set of assumptions about a topic they study (paradigm)
  - this set of assumptions concerns:
    - how knowledge is gathered
    - how the world is organized
  - this in turn results in two dimensions:
    - subjective-objective (wrt knowledge)
    - conflict-order (wrt the world)
  - which results in 4 archetypical approaches to requirements engineering

Conceptual modeling

- **Four approaches to RE:**
  - functional **(objective+order): the analyst is the expert who empirically seeks the truth**
  - social-relativism **(subjective+order): the analyst is a `change agent'. RE is a learning process guided by the analyst**
  - radical-structuralism **(objective+ conflict): there is a struggle between classes; the analyst chooses for either party**
  - neohumanism **(subjective+conflict): the analyst is kind of a social therapist, bringing parties together**

Elicitation techniques

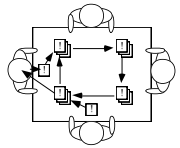
- **Asking:**
  - **interview**
  - **Delphi technique**
  - **brainstorming session**
- **Observing**
  - **task analysis**
  - **scenario analysis**
  - **ethnography**
  - **form analysis**
  - **synthesis from existing system**
- **Others:**
  - **analysis of natural language descriptions**
  - **domain 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

Questions

- Which problems can arise when making explicit the implicit model in conceptual modeling?
- What is the purpose of requirements elicitation?
- Name a number of elicitation techniques?

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

Use-Cases: describing how the user will use the system

- A *use case* is a typical sequence of actions that a user performs in order to complete a given task
  - The objective of *use case analysis* is to model the system from the point of view of
    … how users interact with this system
    … when trying to achieve their objectives.
    It is one of the key activities in requirements analysis
  - A *use case model* consists of
    - a set of use cases
    - an optional description or diagram indicating how they are related

Use cases

- A use case should
  - Cover the *full sequence of steps* from the beginning of a task until the end.
  - Describe the *user's interaction* with the system ...
    - Not the computations the system performs.
  - Be written so as to be as *independent* as possible from any particular user interface design.
  - Only include actions in which the actor interacts with the computer.
    - Not actions a user does manually

Scenarios

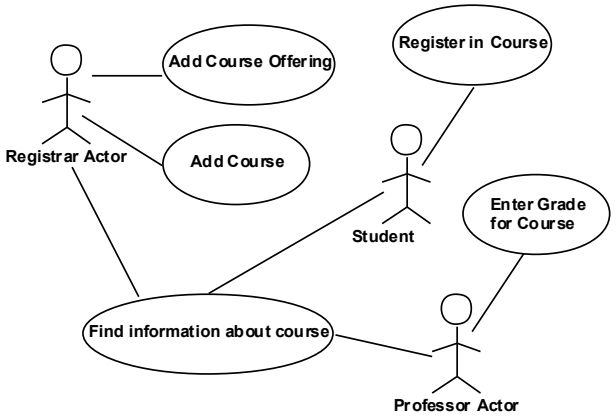
- A scenario is an *instance* of a use case
  - A *specific occurrence* of the use case
    - a specific actor ...
    - at a specific time ...
    - with specific data.

How to describe a single use case

A. **Name**: Give a short, descriptive name to the use case.
B. **Actors**: List the actors who can perform this use case.
C. **Goals**: Explain what the actor or actors are trying to achieve.
D. **Preconditions**: State of the system before the use case.
E. **Summary**: Give a short informal description.
F. **Related use cases.**
G. **Steps**: Describe each step using a 2-column format.
H. **Postconditions**: State of the system in following completion.

- A and G are the most important

Use case diagrams



The modeling processes: Choosing use cases on which to focus

- Often one use case (or a very small number) can be identified as *central* to the system
  - The entire system can be built around this particular use case
- There are other reasons for focusing on particular use cases:
  - Some use cases will represent a high *risk* because for some reason their implementation is problematic
  - Some use cases will have high political or commercial value

The benefits of basing software development on use cases

- They can
  - Help to define the *scope* of the system
  - Be used to *plan* the development process
  - Be used to both develop and validate the requirements
  - Form the basis for the definition of test cases
  - Be used to structure user manuals

Use cases must not be seen as a panacea

- The use cases themselves must be validated
  - Using the requirements validation methods.
- Some aspects of software are not covered by use case analysis.
- Innovative solutions may not be considered.

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)

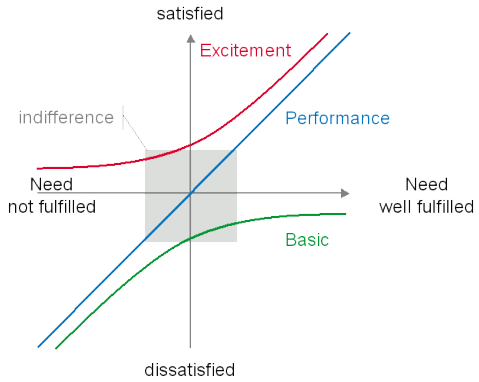
- **Must haves:** top priority requirements
- **Should haves:** highly desirable
- **Could haves:** if time allows
- **Won't haves:** not today

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**
- **…**

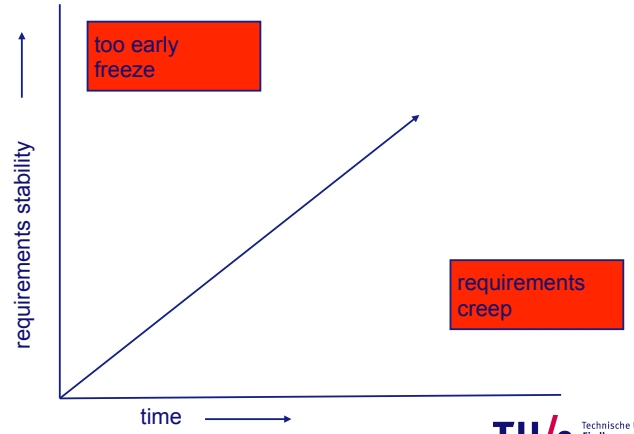
IEEE Standard 830

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



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

The 7 sins of the analyst

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

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

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

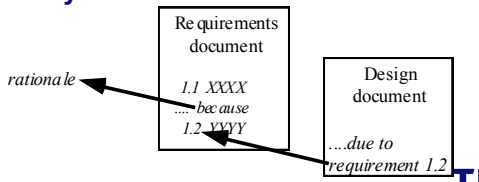
1. Does the (software) product have a succinct name, and a clearly described purpose?
2. Are the characteristics of users and of typical usage mentioned? (No user categories missing.)
3. Are all external interfaces of the software explicitly mentioned? (No interfaces missing.)
4. Does each specific requirement 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***