Chapter 6
Requirements Elicitation II

Organizational Requirements Engineering

Prof. Dr. Armin B. Cremers
Sascha Alda

b-it
Overview of remaining sessions

- 7.12.2005 *no session* (Dies Academicus)
- 14.12.2005 Requirements Analysis and Verification
- 11.1.2006 Non-functional Requirements
- 18.1.2006 Interactive Systems
- 25.1.2006 Practice Talk “Deutsche Post World Net”
- 1.2.2006 Practice Talk “sd&m”
- 8.2.2006 Written Exam
Overview

- General view on requirements elicitation
- Processes of requirements elicitation (and analysis)
- Elicitation Techniques
  - Scenarios
    - Interviews
    - Observation
    - Prototyping
  - From scenarios to use cases
    - Identifying actors
    - Best practice for modeling use cases
    - Refinement of use cases
- Conclusions

This lesson
Interviews Techniques
Summing-up from last lesson

- The most common technique of requirements elicitation.
- During elicitation (early)
  - Understanding role of interviewee within organization
  - Understanding the work context
  - Basis for the creation of first visionary scenarios
  - Reviewing first pass of scenarios (→ exercise)

  **Goal:** Description of complete scenarios

- During analysis
  - Discussing use cases with client and users
  - Correction and refinement (requirements and functionality)

  **Goal:** Getting complete use cases
Interview experiment in ORE Exercise
Suggestion List from Developers

- Allow for different access points (SMS, Voice, Email, WAP)
- Authentication to the SYSTEM (younger people)
- Establish a dictionary explaining common concepts and terms
- Initiation of Maintenance Scenario also by End-User
- Reason about the User-Interface (too many clicks)
- Feasibility of Cooking functionality (Microwave)
- Non-functional Requirements (Costs, Budget, Performance)
- Reusability of components (Pay online use case)
Brainstorming

- Brainstorming refers to the process of systematic and liberal generation of a large volume of ideas from a number of participants.
- Participants are encouraged to provide creative inputs in an atmosphere that is free of criticism and judgment from other participants.
- Unstructured brainstorming
  - Participants can give ideas as these come to mind
  - Quite often not very efficient
- Structured brainstorming
  - Participants must follow in order to make the gathering of inputs more orderly and more efficiently.
Brainstorming
Process of Structured Brainstorming

- State the central brainstorming theme in question form and write it down where every participant can see (white board or flipchart)
  - Ensure that all the members have a full understanding the question
- Let each team member have a turn to give his or her input as answer to the question.
  - If a team member can't think of any input when his or her turn comes, he simply needs to say 'Pass,' and the next member gets the turn.
- Write each input on the board or flipchart as it is given.
  - Nobody is allowed to criticize an input, no matter what.
  - Moderator writes down the input on the board or flipchart using exactly the same words used by the input giver.
- Repeat the brainstorming rounds until everybody says 'Pass' in the same round (→ ideas are exhausted).
- Review each of the listed inputs for further improvement and maximize its clarity.
  - Other team members can ask the input giver what he or she actually means by his or her input.
A Future Workshop is a method that supports the generation of visions for the future use of computers

Origin
- Original application: Citizen participation in urban and land-use planning
- „Deep“ participation within restricted time resources

Participants
- Two facilitators
- Up to 20 Participants

Goal
- Creation of a shared understanding of the mutual problem contexts and the possible solution space
- The outcome of a Future Workshop is a list of desired changes.
Future Workshop II
Three Phases

- **Critique:**
  - Deficits of the current situation, e.g. current work practice, technology support
  - Focus on the negative aspects of an existing system
  - Things that the participants want to change

- **Utopia:**
  - Phantasising about improved situations (new practices, support)
  - Result from the Critique phase is turned into positive ideas.
  - By imagining about how the workplace could change a list of preferred changes is generated.
  - Not restricting ideas by considering available resources!

- **Implementation planning**
  - A collection of the ideas from the Utopia phase is evaluated with respect to how realistic they are.
  - Determine the resources that it would require to implement them.
  - A selection of these ideas are presented as the result of the workshop
Future Workshop II
Techniques and Tools

- Critique
  - Structured brainstorming
  - Contributions from all participants (participants can speak freely)
  - Often useful: Restrict contributions to 30 seconds
  - Visualization/Logging of contribution on cards/stickers
  - Usage of software support (Whiteboards, Anonymous discussion tools)

- Utopia
  - Structured brainstorming
  - Using problem situations/contexts to generate positive visions of the future
  - Re-grouping of contributions according to new future scenarios and refinement in (smaller) work groups

- Implementation planning
  - Presentation of visions from the work groups
  - Planning next steps towards the realization of the visions considering the available resources
Future Workshops IV:
Pros and Cons

- Free formulation of new ideas (+)
- High quality outcome (+)

- Heterogeneous communication skills: Danger of „opinion leadership“ (-)
- Power relations: Managers vs. „ordinary“ staff members (-)
- Very time-consuming (-)
- Scheduling (!)
- Actual Implementation: Will plans/agreements hold? (?-)

Armin B. Cremers, Sascha Alda
Organizational Requirements Engineering
11
Observation

- People often find it hard to describe what they do because it is so natural to them.
- Actual work processes often differ from formal, prescribed processes
  - Sometimes the best way to understand it is to observe them at work
- Approach: adopt methods e.g. from the social sciences which has proved to be valuable in understanding actual work processes
- Approach presented here: Ethnography
Ethnography

- Refers to the qualitative observation and description of human practices in a society or culture (e.g. employee in a bank)
- An ethnographer spends some time observing people at work and building up a picture of how work is done
- Fundamental Theory:
  - The full understanding of a culture emerges *only* when an observer becomes part of it, relates to the people involved and knows the importance of the detailed practices to go on
- Rationale for incorporating Ethnography in the software development process
  - Realization of developers: understanding the domain in which a system is operating is of tremendous importance
  - Software is produced for human beings that have social characters
**Ethnography Guidelines**

**Understanding requirements (Sommerville)**

- Assume that people are good at doing their job and look for non-standard ways of working (resist on individual experience)
- Spend time getting to know the people and establish a trust relationship
- Keep detailed notes of all work practices. Analyze them and draw conclusions from them
- Combine observation with open-ended interviewing
- Organize regular de-briefing session where the ethnographer talks with people outside the process
Ethnography Guidelines

- Problem: Ethnography is a methodology is non-judgmental (no hints for how practices could be improved)
- Integrate other methods to elicit requirements (prototyping)
- Two phases:
  - Analysis: Initial understanding of the system and application domain
  - Focussed Ethnography: discover answers from questions which are raised during prototyping
Viewpoints have been proposed as a mechanism to organize and structure ethnographic records.

Goal: Ethnography must be presented from three viewpoints

The work setting viewpoint
- describes the context and the physical location of the work
- describes how people use objects to carry out tasks

Social and organizational perspectives
- Tries to bring out the day-to-day experience of work as seen by different people who are involved
- Each individual typically sees the work in different ways
- This viewpoint tries to organize and integrate all of these perceptions.

The workflow viewpoint
- presents the work from a series of work activities with information flowing from one activity to another
Ethnography
Pros and Cons

- Methodology is well-funded within social domains (+/−)
- If applied adequately with other methods from Software Engineering, Ethnography delivers an extensive overview of working processes (+)

- Observation processes may last very long (−)
- No standards established (−)
Mock-Ups I

- Representation of computer applications on paper and cardboard visualizing system properties at an very early stage
- Informal way representing details
- Origin: P. Ehn and M. Kyng (UTOPIA project)

- Advantages
  - Support communication among stakeholders
  - Easy to understand
  - Cost-effective
  - Ability to modify setting not dependent on ability to use technology!
Mock-Ups II

- Problems using Mock-ups
  - System behavior can’t be represented well (-)
  - Some (non-presentable) technological restrictions have to be considered (-)
  - Presenting mock-ups suggest easy development (-)
  - Changing the setting may require considerable amounts of time (-)

- Little-old fashioned, rely on using UML models (semi-formal)

- Alternative: Prototypes
Prototyping

- A prototype is an initial version of a system which is available early in the development phase.
- Prototypes are valuable for requirements elicitation because users can experiment with the system and point out its strengths and weaknesses of the implemented requirements.
- Rapid development of prototypes is essential so that they are available early in the elicitation process.
  - Some functionality may be left out.
  - Non-functional requirements (performance) are less stringent.
  - No secondary functions (e.g. maintenance).
  - No complete documentation.
Prototyping benefits

- The prototype allows users to experiment and discover what they really need to support their work
- Establishes feasibility and usefulness before high development costs are incurred
  - Can even reduce the development costs
- Essential for developing the ‘look and feel’ of a user interface
  - Probably the only technique to validate interface requirements
- Can be used for system testing and for further development of documentation
  - Back-to-Back testing: same tests are applied to both prototype and final system
- Forces a detailed study of the requirements which reveals inconsistencies and omissions
Types of prototyping

- **Throw-away prototyping**
  - Intended to help elicit and develop the system requirements.
  - The requirements which should be prototyped are those which cause most difficulties to customers and which are the hardest to understand. Requirements which are well-understood need not be implemented by the prototype.

- **Evolutionary prototyping (Increment)**
  - Intended to deliver a workable system quickly to the customer.
  - Therefore, the requirements which should be supported by the initial versions of this prototype are those which are well-understood and which can deliver useful end-user functionality. It is only after extensive use that poorly understood requirements should be implemented.
Prototyping Costs and Problems

- Training costs: prototype development may require the use of special purpose tools, languages, frameworks and methods
- Development costs: depend on the type of prototype being developed
- Extended development schedules: developing a prototype may extend the schedule
- Incompleteness: it is often not possible to prototype critical system requirements (non-functional requirements)
  - Fact: Use prototype only to discover/discuss functional requirements
- User could be tempted to adopt the prototype directly
- Prototyping interactive systems is much easier than other types of application (real-time or critical systems)
Approaches to prototyping

- **Paper prototyping**
  - a paper mock-up of the system is developed and used for system experiments

- **‘Wizard of Oz’ prototyping**
  - a person simulates the responses of the system in response to some user inputs

- **Executable prototyping**
  - a programming language or other rapid development environment is used to develop an executable prototype
Executable prototype development

- Visual programming languages such as Visual Basic
- Internet-based prototyping solutions
  - HTML browsers
  - Java (Swing, AWT classes for GUI generation)
- Visualization Tools like Microsoft Visio
  - Static elements (GUI-components) and behavior visualization
- Powerful CASE tools
  - Eclipse (Java, with GUI plugin (V3.0))
  - NetBeans (GUI maker is already included and sophisticated)
Overview

- General view on requirements elicitation
- Processes of requirements elicitation (and analysis)
- Elicitation Techniques
  - Scenarios
  - Interviews
  - Observation
  - Prototyping
- From scenarios to use cases
  - Identifying actors
  - Best practice for modeling use cases
  - Refinement of use cases
- Conclusions
Identification of actors

- Recall: Actors represent external entities that interact with the system
- Can be human or an external system
- Quite often correspond to roles (e.g. in an organization)
  - Actors are abstractions of roles

- Is the first step of requirements elicitation
- Serves to define the boundaries of system
- Represent all perspectives from which the developers need to consider the system
- An actor that interact with the system has *goals*
- At some stages it also has *responsibilities*
An actor has goals; goals name use cases; a use case has scenarios naming sub-use cases.
Embracing goals in requirements

- Embracing the goals the system supports for its actors makes good functional requirements. Examples:
  - “Place an order.”
  - “Get money from my bank account.”
  - “Get a quote.”
  - “Find the most attractive alternative.”
  - “Set up an advertising program.”

- Goals summarize system function in understandable, verifiable terms of use.
The use case pulls goal & scenarios together
Each scenario specifies how one condition unfolds.

The use case name is the goal statement:
“Order product from catalog”

Scenario (1): Everything works out well ...
Scenario (2): Insufficient credit ...
Scenario (3): Product out of stock ...

Use case is goal statement plus the scenarios.

Note the grammar: active verb first
Scenario and Use Case

Goal: “Place order” (Use Case)

Sub goal:

(Success scenarios)

(sc1 sc2 sc3 sc4 sc5)

(Failure sc.)

(sc6 sc7 ...
Elicitation Guideline: Identify Actors and Goals

- Identify the actors and their goals and responsibilities.
- Actors should be named with noun phrases (e.g. Officer, Clerk)
- The actors’ goals lead to some first use case descriptions

- Top Down: From initial use case to scenario
- Bottom Up: From Scenarios to use case (abstraction)
- Mixed methods are most practical

**Result**: a list of use cases, a sketch of the system, all actors.
  - Short, fairly complete list of usable system functions.
Elicitation Guideline: Possible Questions to identify actors

- Which user groups are supported by the system to perform their work?
- Which user groups execute the system’s main functionality?
- Which user groups perform secondary functions (maintenance, administration)?
- With what external hardware or software system will the system interact?
- What does each actor need our system to do?
  - Each need shows up as a trigger to the system.
- What are the goals of an actor? Does each actor have a goal?
- What kind of responsibility does actors possess?
- What computers, subsystems and people will drive our system?
  - An “actor” is anything with behavior.
- Which actors can initiate sequences of interaction?
  - Specify access control for the system at a coarse level
Elicitation Guideline: Develop First Scenario

- For each use case write the main success scenario (the happy day scenario).
  - Easiest to read and understand.
  - Understand the user’s assumptions about the system
  - Good Basis for further communication
  - Everything else is a complication on this.

- Result: readable description of system’s main function.
Scenario and Use Case: Specify More Scenarios, derive Use Cases

- Specify a horizontal slice (i.e. many scenarios) to define the scope of the system.
  - Discuss the scope with the user
- Derive more (new) use cases
  - Identify scenarios with common behavior
  - Summarize these into an abstract use case
- Use illustrative prototypes (mock-ups, executables) as visual support
  - User interface design should occur as a separate task after the functionality is sufficiently stable
- Present the user with multiple and different alternatives
- Refine the use cases (Iteration, Increments)
- Result: Broad documentation of the system
Scenario and Use Case: Description of failure conditions

- Identify a complete (?) list of failure conditions
- Describe these as extensions, keep the failure condition separately:
  - Alternate scenarios
  - Use Cases (<<extends>> relationship)
- Types of failures:
  - Recoverable extensions (rejoin the main course later on)
  - Non-recoverable extensions (abort the course of interaction directly)
  - Anticipatable vs. non-anticipatable failures
- Can write “if” statements.
- **Result**: Complete set of use cases covering complete functionality
- **Remark**: 80% of process modeling is about to model exceptions
Scenario and Use Case: Refinement of Use Cases

- The following aspects, initially ignored, are detailed during refinement:
  - Elements that are manipulated are specified by the system (What kind of data is manipulated?)
  - Low-level Sequence of interactions between actor and system (How is a response data produced?)
  - Access rights (which actors can invoke which use cases?)
  - Missing exceptions and handlers
  - Common functionality among use cases are factored out (use <<include>> relationship, generalization)

- **Result**: complete and consistent set of use cases ready for further analysis
Use cases should be named with verb phrases. The name should indicate the goal the users tries to accomplish

- Active Verbs (ReportAccident, OrderBook) for normal use cases
- Passive Verbs (BookIsSoldOut, AccidentIsAlreadyReported) for extending use case (<<extends>>) (not always easy)

Use case steps in the flow of event should be phrased in active speed

- Makes it explicit who accomplished the step

Causality relationship between steps must be kept

- Which cause precedes the effect?

A use case has to describe the entire user transaction

A use case should not exceed 2-3 pages

- Otherwise, use <<include>> and <<extends>> relationship

A use case should not describe the user interface of a system
Scenario and Use Case: Identify Analysis Objects (Glossary)

- Identify participating objects for each use case (and scenarios) to establish a clear terminology between developer and user
- Describe these unambiguously and collate them into a glossary
- Result: Consistent set of definitions

- Only a first object model for discussion purposes
- Serves an input for a more detailed analysis model (not discussed with customer)
Scenario and Use Case: Identify Analysis Objects (Glossary) - 2

- Heuristics for identifying initial analysis objects
  - Terms that developers and users must clarify to understand use cases
  - Recurring nouns in the use cases (e.g. Report, Ticket, Book)
  - Real-world entities that the system must track (e.g. Person, Resource)
  - Processes, Use Cases (OrderBook, ReportIncident)
  - Data sources and sink (e.g. Database, Printer)

- Heuristics for cross-checking objects and use cases
  - Which use case creates this object?
  - During which use cases are the values of the object attributes entered?
  - Which use case destroy an object?
  - Is an object actually needed?
Non-functional Requirements:

- Non-functional requirements are defined at the same time as functional requirements because they have similar impact on the development and costs of the system.
- Use cases provide no ways for describing non-functional requirements.
  - Best practice: Embrace them in the textual representation (Quality requirements)
- Outlook: separate session about non-functional requirements
Non-functional Requirements: Some possible questions

- **Performance characteristics**
  - Are there any speed, throughput, or response time constraints on the system?
  - Are there size or capacity constraints on the data to be processed by the system?

- **Error handling and extreme conditions**
  - How should the system respond to input errors?
  - How should the system respond to extreme conditions?
Summary

- Eliciting requirements is a creative act (developers as well as users)
- There is no one solution that fits all problems
  - 1. Different techniques are to be used for different problems
  - 2. Techniques are toolboxes that have to be fit in the special problem
- Requirements elicitation is a learning process for users and developers