

## 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

## 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

## The modeling processes: Choose 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

- Use cases have shortcomings:
  - 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.

## Some Techniques for Gathering and Analysing Requirements

- Observation
  - Read documents and discuss requirements with users
  - Shadowing important potential users as they do their work
  - Session videotaping
- 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

## Gathering and Analysing Requirements

- 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

## Gathering and Analysing Requirements

- 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

## Level of detail required in a requirements document

- How much detail should be provided depends on:
  - 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

## 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

## Requirements Review Checklist

1. Does each user requirement have a **unique identifier**?
2. Is each user requirement **atomic** and **simply formulated**? (Single sentence. Composite requirements can best be split.)
3. Are user requirements organized into **coherent groups**? (If necessary, hierarchical; not more than about ten per group.)
4. Is each user requirement **verifiable** (in a provisional acceptance test)? (Where possible, quantify.)
5. Is each user requirement **prioritized**?
6. Are all **unstable** user requirements marked as such? (TBC='To Be Confirmed')

## Requirements Review Checklist (cont.)

7. Are all user requirements **necessary**?
8. Are the user 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.)
9. Are the user requirements **consistent**? (Non-conflicting.)
10. Are the user requirements sufficiently **precise** and **unambiguous**? (Which interfaces are involved, who has the initiative, who supplies what data; avoid passive voice.)
11. Are the user requirements **understandable** to those who will need to work with them later?
12. Are the user requirements **realizable** within budget?

## Requirements Review Checklist (cont.)

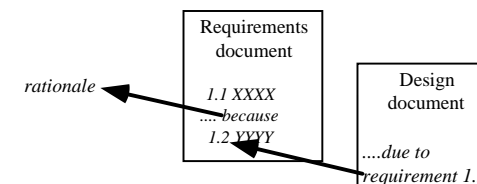
13. Do the user requirements express actual **customer needs** (in the language of the problem domain), **rather than solutions** (in developer jargon)?
14. Are the user requirements not **overly restrictive**? (Allow enough developer freedom.)
15. Is **overlap** among user requirements pointed out explicitly? (Redundancy; cross-reference.)
16. Are **dependencies** between user requirements pointed out explicitly? Are these consistent with the priorities?
17. Are the characteristics of **users** and of **typical usage** described in sufficient detail? (No user categories missing.)

## Requirements Review Checklist (cont.)

18. Is the **operational environment** described in sufficient detail, including e.g. relevant data flows in the "outside" world that do not interact directly with the system? (No external interfaces missing. No capabilities missing. Diagram included.)

## Requirements documents

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



## Managing Changing Requirements

- Requirements change because:
  - Business process changes
  - Technology changes
  - The problem becomes better understood

## Managing Changing Requirements

- 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*