Writing Effective Requirements Specifications
Properties

• Complete
  – All relevant scenarios must be covered
  – Forgetting requirements is expensive

• Realistic
  – Functional requirements and non-functional requirements should not contradict each other
    • e.g.: the response time of a system which solves mixed integer programming problems is usually long
Properties

• Correct
  – The requirements must correctly reflect the challenges posed by the environment
    • Negative example: Y2K-Problem

• Modifyable
  – Related requirements should be grouped together, so that changes can be made locally

• Ranked according to priority
Properties

• Verifiable
  – Make sure that requirements can be checked objectively
    • Include measures
      – Transaction/sec
      – MTBF
      – ...
    • Subjective: The system should be easy to use
    • More objective: An experienced user should be able to use the system after two hours training
Properties

• Traceable
  – Each requirement must be uniquely identified

• Unambiguous
  – Specifies in a concise language which does not allow alternative interpretations (difficult to achieve)

• Valid
  – Understood and accepted by all project members, managers, customers involved
Recommendations

• Language
  – Choose imperatives carefully. Distinguish
    • shall: describes required system functionality
    • must, must not: describes a constraint
    • should: suggests functionality
  – Reduce ambiguity
    • avoid options: can, may, optionally
    • avoid weak phrases: as a minimum, as appropriate, easy, adequate
Recommendations

• Language
  – Use readable, simple language
    • short sentences
    • generally understood words
  – Decompose long requirements into parts
Recommendations

• Elements of a good requirements statement
  – Localization/scenario, e.g. “In online mode”
  – Actor/owner, e.g. “the static tolerance band agent”
  – Action, e.g. “maintains a log of limit violations”
  – Target/owned, e.g. “for the selected signals”
  – Constraint, e.g. “provided the appropriate tolerance band have been defined by the plant administrator”

• Within a use case, Localization/Scenario, is provided by the context
Recommendations

• Documentation Standard
  – Minimize general and administrative sections in your documents. The requirements should be the largest part.
  – Use templates, but
    • customize them to the needs of your projects (omit useless sections)
    • don’t invent meaningless texts to fill all sections in the template
Recommendations

• Documentation Standard
  – Number all requirements
    • Make sure (e.g. using tool support), that
      – the numbering scheme is applied consistently in all documents
      – every requirement has a unique number
    • Within a use case, denote each step on a separate, numbered line
  – When using examples, illustrations, tables
    • Mark them uniquely
    • Explain their purpose (“This is an example for ...”) and structure (“Column1 describes ...”)
Recommendations

• Choose the right granularity
  – Since requirements are to be read by humans, a use case with 100 steps might be too long

• Conduct reviews
  – Review quality of content
  – (Separately) Review conformance with guidelines and standards
Use and Abuse of Use Cases

• Use case advantages
  – Capture a user’s need
  – Input to the testing process
  – Unit of work for incremental development
Abuse of use cases

• Abuse by decomposition
  – Many designers use <<uses>> relationships among use cases for functional decomposition
    
    function
    \rightarrow subfunction
    \rightarrow subsubfunction
  
  – Problems
    • Contradicts the OO style
    • Subsubfunctions are duplicated (under different functions)
    • Objects are only context-specific encapsulations of data in this approach
  
  – Do not try to design the program using use cases
    => leave out detail
Abuse of use cases

• By abstraction
  – Use cases are intended for communication.
  – There is no need to abstract from the concrete use cases, even if the implementation will do so.
  – The abstraction might not be natural. Time is lost by discussing it.
  – If you abstract from “Send receipt to customer” to “Transmit or generate document for stakeholder”, you will have a large use case, which will be hard to understand and implement
=> “Use the concrete use cases to explain and verify your powerful abstractions.”
Abuse of use cases

• By GUI
  – Today’s GUI builders allow to describe use cases via GUI prototypes
  – Problems
    • The user thinks, that everything is done, when he sees the GUI prototype => false indication of progress
    • The user will not accept later changes to the GUI easily
Abuse of use cases

• By denying choice
  – Use cases should really describe goals, i.e. problems the user would like to solve.
  – Often one tends to commit to early to describing a solution; this keeps us from considering alternative solutions
  – Example
    • apply style (in Word) <-> Format paragraph