

Quality of Requirements: Consistency

- Consistency
 - no contradictions
 - the program won't work if requirements are contradictory
- Hard to guarantee
 - large sets of requirements
 - contradictions can be hidden
 - example:
 - Section 1.2 says: “no more than 128MB of memory needed”
 - Section 5.8.9 says: “images should be rendered in real time”
 - => is this a contradiction???
- Formal logic: anything follows from a contradiction
- Problem:
 - often difficult to explain contradiction to customers
 - customers may want impossible things

Quality of Requirements: Manageability

- Resources must match requirements
 - can it be done in time?
 - with the money available?
 - with the skills we have?
- Risk management
 - requirements should be prioritized
 - ideally: alternatives in case it doesn't work out
 - often impossible to tell which requirements are possible
- Complexity management
 - don't do everything at once
 - iterative processes
 - prototyping

Quality of Requirements: Specificity

- Be as precise and detailed as possible
- Bad:
 - “program shall be fast”
 - “it takes a number as input”
- Good:
 - “the program shall give a response in less than 1s”
 - “it takes a 16-bit signed integer as input”
- Definitions
 - are often a good idea
 - example: “by 'Number', we always mean a 16-bit signed integer”
 - defined terms are often capitalized
- Rules about definitions
 - no circles

Quality of Requirements: No Implementation Bias

- Implementation bias:
 - giving information about the design
 - giving information about the code
- Use terminology of the domain
 - not technical terminology
 - you want to focus on WHAT
 - leave HOW for later
- Bad examples:
 - “store the checked-out books in an array”
 - “calculate the square root with Newton's algorithm”
- Good examples:
 - “the library knows which books are checked out”
 - “return the square root with 5-digit precision”

Nonfunctional Requirements

- Software Qualities
- External versus internal
 - external = for the user
 - internal = for the developer
- Product quality versus process quality
- Advice
 - be specific
 - prioritize
 - which are relevant for your project?
 - how to verify them?

Software Qualities

- Correctness
 - mathematical property
 - can be proven
 - in theory
 - based on specification
- Reliability
 - statistical
 - depends on operational profile
 - seriousness of error considered
- Robustness
 - failing gracefully
 - appropriate error messages
 - error recovery

Software Qualities

- Performance
 - time
 - space
 - algorithmic complexity
 - depends on hardware, OS,...
- Usability
 - easy to use
 - learnability
 - different for different users
 - newbies
 - experienced users

Software Qualities

- Verifiability
 - can be tested
 - can be understood
- Maintainability
 - repairability
 - evolvability
- Reusability
 - procedures / functions
 - libraries
 - components
- Portability
- Interoperability

Specialized Qualities: Distributed Systems

- Interoperability in the network
 - network protocols
 - platforms, users
- Scalability
 - works with many users
- Response time
 - similar to performance
 - perceived performance vs. actual performance
 - example: HTTP
- Distribution
 - client-server: centralized
 - no single point of failure
 - peer-to-peer

How to Validate Requirements?

- Validation
 - Making sure the requirements are right
 - desired
 - consistent
 - complete
- Discussing them with users
 - Showing them mockups, use cases, feature lists
 - Explaining them to the user
- Brainstorming
 - Is anything missing?
 - Creativity needed
- Role Playing, Walkthroughs
 - Developers play users
 - Go through use cases step-by-step, using mockups