Properties of Good Requirements

(Nonexhaustive) List

- Understandable by end users
- Nonprescriptive
- Correct
- Complete
- Concise (succinct)
- Precise
- Clear
- Unambiguous
- Consistent
- Traceable
- Modifiable
- Testable (verifiable)
- Feasible

Understandable by End Users

- Requirements as contract
- Consider the requirement

  "To ensure predictable operation, the system shall not employ nondeterministic methods."

  - we think "race conditions during concurrency"
  - users mostly have no notion of these things

  - Primary requirement: "System operation shall be predictable and repeatable."

    - Derived requirement: "The system shall not employ nondeterministic methods."

      » a design constraint

Nonprescriptive

- Requirements about domain
  - characteristics of the software that can be observed externally
- Consider the following:

  "The software shall employ B-trees for storage of information kept in memory."
  - does not describe the behavior of the system externally
  - rule of thumb: algorithms and data structures belong in design
    - design suggestions should be documented though
    - design constraints are possible from the customer

Correct

- The user is the sole judge of correctness (scary!)
  - user intent must be represented accurately
  - conflicts in user intent are the subject of requirements analysis
- Consider

  "The system shall accept operator input from up to and including 29 consoles."
  - customer might want to connect more to the system later
  - customer might actually intend to connect only five
    - in either case, the results of magic number 29 can be problematic
      - » bad result if number should be larger
      - » wasted effort if number should really be smaller
  - developer option: detail tradeoffs to customer, encourage accuracy

Complete

- Two aspects
  - "missing" requirements
  - information missing from an individual requirement
    - Consider the second type of problem:

    "The system shall provide the operator with the information needed to safely shut down the controlled machinery when an exception occurs."
      - a "blanket" requirement trying to cover a lot of ground
    - Maybe this is all that is needed:

    "The system shall provide the operator with time-stamped messages describing system exceptions."
      - operator actions and safety were meant to be covered elsewhere in training and operator manuals.

- Sometimes full "completeness" is not possible
  - put in the all-important placeholder to assure consideration!

    "The system shall provide the operator with time-stamped messages describing system exceptions (list of exceptions **TBD**, **refs.**) The messages shall not lag more than **TBD (refs.)** seconds behind the exceptions they describe."

    - The location, subject, and any known references for the TBD are themselves important information to record.

Concise (Succinct)

- No rambling prose, compound sentences with extraneous information

  "We feel that good systems provide the end user with good value. Because of this, we think that the system should provide adequate performance with a 2 Gb disk, since this is the least expensive disk that we may purchase from the designated vendor. Of course, the user may elect to configure the system with a larger disk, and we recommend this, but we have attempted to come to grips with most of the problems raised by use of the smaller disk, and we feel that they can be, by and large, satisfactorily resolved."

  - aaaargh. Try this instead:

    The system shall fulfill all specified functions when configured with a 2 gigabyte disk."

    » want to put the pertinent information up front

Precise

- Consider

  "The system shall accept valid employee ID numbers from 1 to 9999."

  - All numbers from 1 to 9999 *valid*? (What is "valid" apart from the range?)
  - Is "1" valid? Is "9999" valid?
  - Can we use leading "0's" to get a new ID? (1, 01, 001)
    - are they the same? (what is the length of an ID?)

  – Now consider:

  "The system shall accept only valid ID numbers as defined elsewhere (ref.). No otherwise valid number will be accepted unless it is an integer between 1 and 9999 inclusive, represented without leading zeroes."

    - So "02" is not accepted, nor is 734 if it is not valid (defined elsewhere)

Clear

- Consider

  "The items in tab-separated columns and underscore-separated rows of the output may refer to each other, but no item in (row,column) position (i,j) may refer to another in position (p,q) unless p<i, or if i=p, q<j."

  - not suitable for a general audience
  - write requirements in short, declarative sentences, best at elementary level

    "The output consists of rows and columns. Items across each row are separated by tabs. There is an underscore between rows. When item X refers to item Y, Y must either be in a row above X, or fi they are in the same row, Y must be in a column to the left of X. An item may not refer to itself."

    - examples are helpful, but not useful as requirements
    - pictures may be very useful to illustrate the concepts
    - must *capture the idea itself*, not some examples of it

Unambiguous

- Requirements must be testable! (A good way to find ambiguity...)
  - watch for use of pronouns (refer to the specific noun phrase over and over, even if it sounds a little stilted.)
    - "it," "this"
  - watch for differing interpretations of a requirement
  - other common ambiguities come from the "separator/terminator" distinction
    - cover all cases explicitly

Consistent

- Can arise from systems so complex that no one person can envision the whole thing
- Can also arise from changes in one part of a document not reflected in another part that is not changed

  "The system shall track detected airborne objects traveling at speeds from 200 to 400 miles per hour inclusive."

  many pages later...

  "The system shall flag all detected airborne objects traveling at speeds from 300 to 500 miles per hour inclusive."

  - This *may* be OK if the system is to **flag** some objects it is not required to **track**

Traceable

- Requirements documents guide *all subsequent development*
  - must be able to connect requirements to the rest of development:

  Uniquely identify each part of the requirements document
    - need to be able to refer to specific parts of the document
      - for others over the phone, email
      - to connect to others by reference
      - to trace back when design or implementation requires a change

Modifiable

- May be avoided by keeping key information in once place (not scattered through the document)
  - the data dictionary provides this
- Consider the example of inconsistency:

  "[4.1.5] The system shall track detected airborne objects traveling at speeds from 200 to 400 miles per hour inclusive."

  and

  "[8.1.6] The system shall flag all detected airborne objects traveling at speeds from 300 to 500 miles per hour inclusive."

  - Suppose the inconsistency arose by replacement of "500" in 4.1.5 with "400" and neglecting to change 8.1.6

Modifiable (cont'd)

- Keep key information in one place:

  "[1.1.1] Normal operational range is 200 to 500 miles per hour, inclusive."

  "[1.1.2] Airborne objects are called "exceptional" if they are traveling at speeds of more than 100 miles per hour above the floor of the normal operational range."

  .....

  "[4.1.5] The system shall only track airborne objects traveling in the normal operational range."

  .....

  "[8.1.6] The system shall flag all exceptional objects."

  – the change of the upper limit to 400 in 1.1.1 now causes no inconsistency.

Testable (Verifiable)

- Consider:

  "The software shall operate on a PC for 5000 hours without failure."

  - good luck, testers!
    - they can't test every possible possible set of PC circumstances for 5000 hours
    - narrowing to "standard PC configuration" doesn't help much

  – We need to quantify such requirements to make them measurable and testable.
    - write a simple test plan at the requirements stage!

Feasible

- Requirements must provide a sound basis for design.
- Consider:

  "[1.1] The software shall operate on a 100 megahertz 486 system."

  [1.2] The software shall respond to any critical event within 1 picosecond."

  - hardly!
- Consider what occurred during a class assignment:

  "P9. The required software is a C main program that links with an arbitrary collection of user-supplied subroutines."

  .....

  "P34. An error message will be issued if the main program should try to call a nonexistent user subroutine."

  - but a main program that attempts to link with nonexistent user subroutines cannot be successfully linked (missing entry point), so it can't issue an error message!