How to write better requirements

Rolfe Bozier

6-Feb-2014

Why bother with requirements?

- Requirements in CiSRA can serve 3 purposes:
  - tell the developer what to design and implement
  - tell Canon what to expect (perhaps with their agreement)
  - tell V&V *what* to test against (and *how*)
- CiSRA is generally not requirements-driven, but having clear requirements can help prevent:
  - doing too much developing
  - not doing enough development
  - being embarrassed when it comes time for testing (or delivery!)

Other benefits

- 15 years ago, Intel moved to a framework of better-defined requirements.
  - In the first year, they reduced defects by 50%
- Better requirements can mean the difference between
  - a throwaway document, and
  - a useful reference

What is a requirement?

- Types of requirements:
  1. A statement about what a system must do
  2. A limitation or constraint on design or resources
  3. How well a system should behave
- 1 is a *Functional Requirement*
  - functional requirements specify the operation and results of system actions
- 2 and 3 are *Non-Functional Requirements*
  - non-functional requirements cover the rest: performance, quality, availability, scalability, reliability (the "-ilities")
- Generally, Functional Requirements are easy to define, the difficulty arises with Non-Functional Requirements

4 rules for better requirements

- Rule 1: be precise about the strength
- Rule 2: watch out for incomplete requirements
- Rule 3: watch out for tricky words and phrases
- Rule 4: the requirement must be verifiable

Rule 1: be precise about the strength

- Use the following critical words correctly:

Word	Meaning
“must”	This is an absolute requirement
“must not”	This is an absolute prohibition
“should”	A non-mandatory, but recommended, requirement
“may”	A non-mandatory, truly optional, requirement
“shall”	Do not use (or treat as a synonym for “must”)
“will”, “can”	Do not use (these are statements of fact)

Rule 2: watch out for incomplete requirements

- Incomplete requirements can take several forms:
  - Weak words
    - "fast", "quickly", "reliable", "normal", "secure", "immediately", etc.
      - *how quickly? how fast? what is normal?*
  - Unbounded lists
    - "such as ...", "including ...", "at least ...", "...or later", etc.
      - *should the developer try to exceed them?*
      - *what happens if the list changes in the future?*

Rule 2: watch out for incomplete requirements

- Implicit collections
  - a reference to a set that is not fully defined, e.g.
    - "must run on Linux" (*which versions?*)
    - "must meet Canon performance criteria" (*which ones?*)
- Ambiguity
  - vagueness ("must conform to current standards")
  - subjectivity ("maintain high quality output")
  - optionality ("must render as many objects as possible")
  - under-specification ("must support PCL input")
  - under-reference ("must be compatible with previous releases")

Rule 3: watch out for tricky words and phrases

- Keep your sentence simple and short
  - Your audience may not be a native English speaker
  - Or worse, a software engineer…
- Words with multiple or non-technical meanings
  - “support”, “enable”, “allow”, “provide”, etc.
- Qualifiers
  - “each” refers to one item at a time
  - “every” refers to all of them
- Grammatical ambiguity
  - “must render anti-aliased text and thin lines”
    - *should thin lines be anti-aliased?*

Rule 4: the requirement must be verifiable

- This is the common-sense test
- Read the requirement and put on your testing hat...
  - is it *possible* to verify each requirement?
  - is it clear *how* it should be verified
- Functional requirements
  - usually boolean conditions which are either met or not
- Non-functional requirements
  - usually involve a **scale** (e.g. seconds) and a **meter** ("cpu time")
  - can have up to 3 values: minimum, target, stretch

Let’s look at some examples!

- “Once printing has started, stopping the print head should be avoided”
- “The time between receiving a PDL file and the printer starting should be kept to a minimum”
- “Shall be capable of supporting various mixes of the above tasks”
- “The product shall support cancel functionality”
- “Data structures shall be arranged to be efficient for both 32-byte and 64-byte cache line sizes”
- “Serialisation time should be insignificant compared to build time”

Let's look at some examples!

- "Once printing has started, stopping the print head should be avoided"
- "The time between receiving a PDL file and the printer starting should be kept to a minimum"
- "Shall be capable of supporting various mixes of the above tasks"
- "The product shall support cancel functionality"
- "Data structures shall be arranged to be efficient for both 32-byte and 64-byte cache line sizes"
- "Serialisation time should be insignificant compared to build time"

Let's look at some more examples!

- "The SDK must produce either 8 or 16-bit output per colour channel"
- "Requires output module to provide API to ship N scanlines"
- "Support data structure serialisation"
- "The system must output valid UDI objects in all supported configurations"
- "Mean execution time must be improved (reduced) by any idioms used in the [external] system, and should not be worsened (increased) unnecessarily by the new algorithm"

Let's look at some more examples!

- "The SDK must produce either 8 or 16-bit output per colour channel"
- "Requires output module to provide API to ship N scan-lines"
- "Support data structure serialisation"
- "The system must output valid UDI objects in all supported configurations"
- "Mean execution time must be improved (reduced) by any idioms used in the [external] system, and should not be worsened (increased) unnecessarily by the new algorithm"