



Lecture 3, Part 1: Requirements Specifications

⇒ Why we need to write specifications

- ↳ Purpose and audience
- ↳ Choosing an appropriate size and formality

⇒ Desiderata for Specifications

- ↳ Properties of good specifications
- ↳ Typical problems
- ↳ What not to include

⇒ Structure of a requirements document

- ↳ IEEE standard



Software Requirements Specification

⇒ How do we communicate the Requirements to others?

- ↳ It is common practice to capture them in an SRS
 - But an SRS doesn't need to be a single paper document...

⇒ Purpose

- ↳ Communicates an understanding of the requirements
 - explains both the application domain and the system to be developed
- ↳ Contractual
 - May be legally binding!
 - Expresses agreement and a commitment
- ↳ Baseline for evaluating subsequent products
 - supports system testing, verification and validation
 - enough information to verify whether delivered system meets requirements
- ↳ Baseline for change control
 - requirements change, software evolves

⇒ Audience

- ↳ Users, Purchasers
 - Most interested in system requirements
 - Not generally interested in detailed software requirements
- ↳ Systems Analysts, Requirements Analysts
 - Write various specifications that inter-relate
- ↳ Developers, Programmers
 - Have to implement the requirements
- ↳ Testers
 - Determine that the requirements have been met
- ↳ Project Managers
 - Measure and control the analysis and development processes



Appropriate Specification

⇒ Consider two different projects:

- A) **Tiny project, 1 programmer, 2 months work**
programmer talks to customer, then writes up a 5-page memo
- B) **Large project, 50 programmers, 2 years work**
team of analysts model the requirements, then document them in a 500-page SRS

	<i>Project A</i>	<i>Project B</i>
<i>Purpose of spec?</i>	Crystalizes programmer's understanding; feedback to customer	Build-to document; must contain enough detail for all the programmers
<i>Management view?</i>	Spec is irrelevant; have already allocated resources	Will use the spec to estimate resource needs and plan the development
<i>Readers?</i>	Primary: Spec author; Secondary: Customer	Primary: programmers, testers, managers; Secondary: customers



A complication: Procurement

⇒ An 'SRS' may be written by...

- ↳ ...the procurer:
 - SRS is really a call for proposals
 - Must be general enough to yield a good selection of bids...
 - ...and specific enough to exclude unreasonable bids
- ↳ ...the bidders:
 - SRS is a proposal to implement a system to meet the CFP
 - must be specific enough to demonstrate feasibility and technical competence
 - ...and general enough to avoid over-commitment
- ↳ ...the selected developer:
 - reflects the developer's understanding of the customers needs
 - forms the basis for evaluation of contractual performance
- ↳ ...or by an independent RE contractor!

⇒ Choice over what point to compete the contract

- ↳ **Early (conceptual stage)**
 - can only evaluate bids on apparent competence & ability
- ↳ **Late (detailed specification stage)**
 - more work for procurer; appropriate RE expertise may not be available in-house
- ↳ IEEE Standard recommends SRS jointly developed by procurer & developer



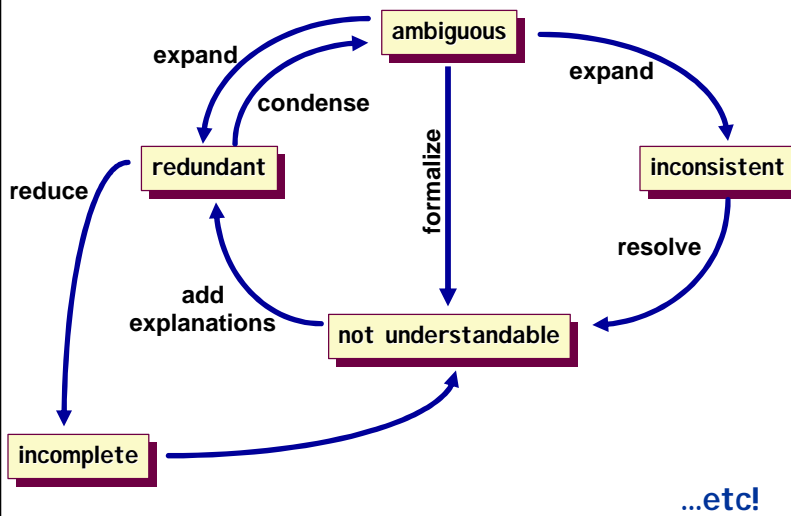
Desiderata for Specifications

Source: Adapted from IEEE-STD-830-1998

- ⊃ Valid (or "correct")
 - ↳ Expresses only the real needs of the stakeholders (customers, users,...)
 - ↳ Doesn't contain anything that isn't "required"
- ⊃ Unambiguous
 - ↳ Every statement can be read in exactly one way
- ⊃ Complete
 - ↳ Specifies all the things the system must do...
 - > ...and all the things it must not do!
 - ↳ Conceptual Completeness
 - > E.g. responses to all classes of input
 - ↳ Structural Completeness
 - > E.g. no TBDs!!!
- ⊃ Understandable (Clear)
 - ↳ E.g. by non-computer specialists
- ⊃ Consistent
 - ↳ Doesn't contradict itself
 - > I.e. is satisfiable
 - ↳ Uses all terms consistently
- ⊃ Ranked
 - ↳ Must indicate the importance and/or stability of each requirement
- ⊃ Verifiable
 - ↳ A process exists to test satisfaction of each requirement
 - > "every requirement is specified behaviorally"
- ⊃ Modifiable
 - ↳ Can be changed without difficulty
 - > Good structure and cross-referencing
- ⊃ Traceable
 - ↳ Origin of each requirement is clear
 - ↳ Facilitates referencing of requirements in future documentation



There is no Perfect SRS!





SRS Contents

⇒ Software Requirements Specification should address:

- ↳ **Functionality.**
 - What is the software supposed to do?
- ↳ **External interfaces.**
 - How does the software interact with people, the system's hardware, other hardware, and other software?
- ↳ **Performance.**
 - What is the speed, availability, response time, recovery time of various software functions, and so on?
- ↳ **Attributes.**
 - What are the portability, correctness, maintainability, security, and other considerations?
- ↳ **Design constraints imposed on an implementation.**
 - Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) and so on?



SRS should not include...

- ⇒ **Project development plans**
 - cost, staffing, schedules, methods, tools, etc
 - ↳ Lifetime of SRS is until the software is made obsolete
 - ↳ Lifetime of development plans is much shorter
- ⇒ **Product assurance plans**
 - CM, V&V, test, QA, etc
 - ↳ Different audiences
 - ↳ Different lifetimes
- ⇒ **Designs**
 - ↳ Requirements and designs have different audiences
 - ↳ Analysis and design are different areas of expertise
 - I.e. requirements analysts shouldn't do design!
 - ↳ *Except where application domain constrains the design*
 - e.g. limited communication between different subsystems for security reasons.



Typical mistakes

- ↳ **Noise**
 - text that carries no relevant information to any feature of the problem.
- ↳ **Silence**
 - a feature that is not covered by any text.
- ↳ **Over-specification**
 - text that describes a feature of the solution, rather than the problem.
- ↳ **Contradiction**
 - text that defines a single feature in a number of incompatible ways.
- ↳ **Ambiguity**
 - text that can be interpreted in at least two different ways.
- ↳ **Forward reference**
 - text that refers to a terms or features yet to be defined.
- ↳ **Wishful thinking**
 - text that defines a feature that cannot possibly be validated.
- ↳ **Jigsaw puzzles**
 - distributing key information across a document and then cross-referencing
- ↳ **Duckspeak requirements**
 - Requirements that are only there to conform to standards
- ↳ **Unnecessary invention of terminology**
 - E.g. 'user input presentation function'
 - E.g. 'airplane reservation data validation function'
- ↳ **Inconsistent terminology**
 - Inventing and then changing terminology
- ↳ **Putting the onus on the development staff**
 - i.e. making the reader work hard to decipher the intent
- ↳ **Writing for the hostile reader**
 - There are fewer of these than friendly readers



Use Appropriate Notations

⇒ Natural Language?

↳ "The system shall report to the operator all faults that originate in critical functions or that occur during execution of a critical sequence and for which there is no fault recovery response."
(this is adapted from a real NASA spec for the international space station)

⇒ Or a decision table?

Originate in critical functions	F	T	F	T	F	T	F	T
Occur during critical sequence	F	F	T	T	F	F	T	T
No fault recovery response	F	F	F	F	T	T	T	T
Report to operator?								



Requirements Traceability

⇒ Definition (DOD-STD-2167A):

- “(1) The document in question contains or implements all applicable stipulations in the predecessor document
- (2) a given term, acronym, or abbreviation means the same thing in all documents
- (3) a given item or concept is referred to by the same name or description in the documents
- (4) all material in the successor document has its basis in the predecessor document, that is, no untraceable material has been introduced
- (5) the two documents do not contradict one another”

⇒ In short:

- ↪ A demonstration of completeness, necessity and consistency
- ↪ a clear allocation/flowdown path (down through the document hierarchy)
- ↪ a clear derivation path (up through the document hierarchy)



Organizing the Requirements

⇒ Need a logical organization for the document

- ↪ IEEE standard offers different templates

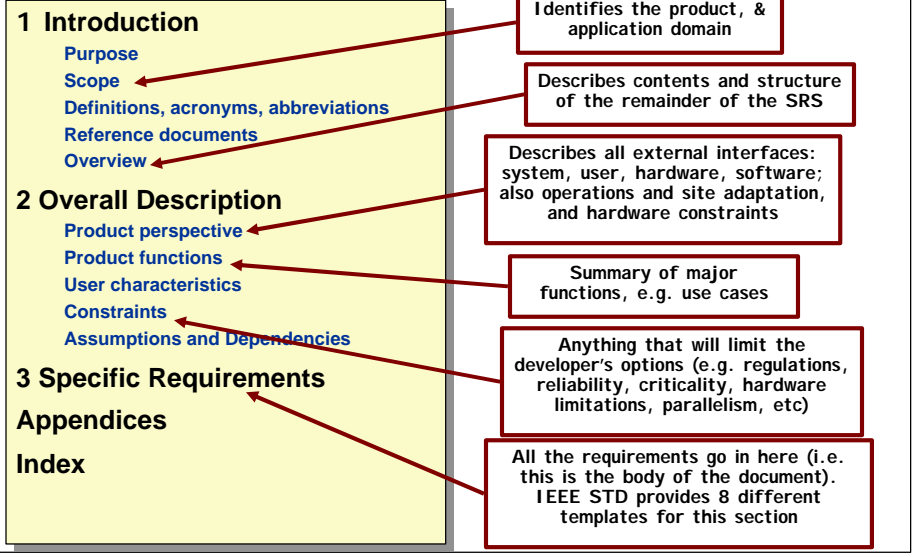
⇒ Example Structures - organize by...

- ↪ ...External stimulus or external situation
 - e.g., for an aircraft landing system, each different type of landing situation: wind gusts, no fuel, short runway, etc
- ↪ ...System feature
 - e.g., for a telephone system: call forwarding, call blocking, conference call, etc
- ↪ ...System response
 - e.g., for a payroll system: generate pay-cheques, report costs, print tax info;
- ↪ ...External object
 - e.g. for a library information system, organize by book type
- ↪ ...User type
 - e.g. for a project support system: manager, technical staff, administrator, etc.
- ↪ ...Mode
 - e.g. for word processor: page layout mode, outline mode, text editing mode, etc
- ↪ ...Subsystem
 - e.g. for spacecraft: command&control, data handling, comms, instruments, etc.



IEEE Standard for SRS

Source: Adapted from IEEE-STD-830-1993. See also, Blum 1992, p160



IEEE STD Section 3 (example)

Source: Adapted from IEEE-STD-830-1993. See also, Blum 1992, p160

