Course 2022-2023

Software Engineering

“Probadlo todo y quedaos con lo bueno” (Pablo de Tarso)

Gonzalo Génova
Jose María Alvarez-Rodríguez
Juan Miguel Gómez-Berbís

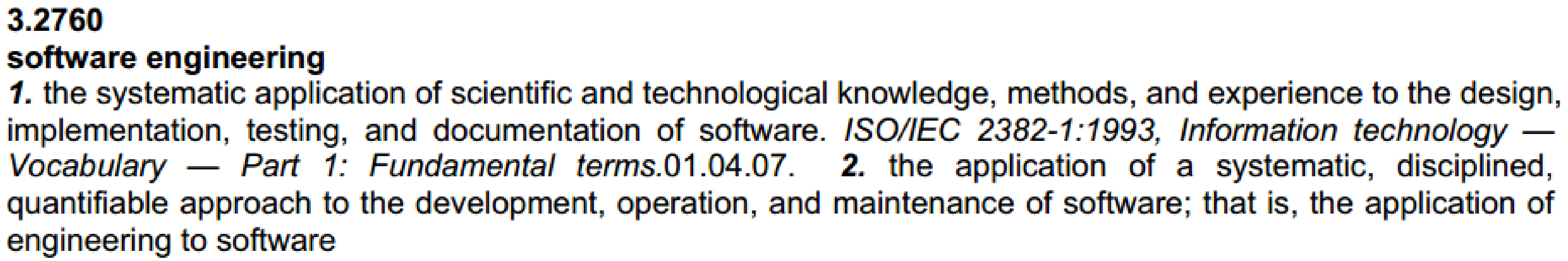
Contents

Introduction

- What is Requirements Engineering?
  - Elicitation and Analysis
  - Definition of requirement
  - The requirements specification document
- The need of requirements engineering
  - To build “something” we need to understand what is “something”
- Why is it necessary to write good requirements?
  - If we do not have requirements is not possible...
  - There is no professional engineering without written requirements
- Problems and issues in requirements engineering
  - The cost is a necessity not an unnecessary luxury
- Requirements engineering in the context of a project
  - Initial planning: a kind of agreement, viability, planning, estimation...

Software Engineering

- Software Engineering

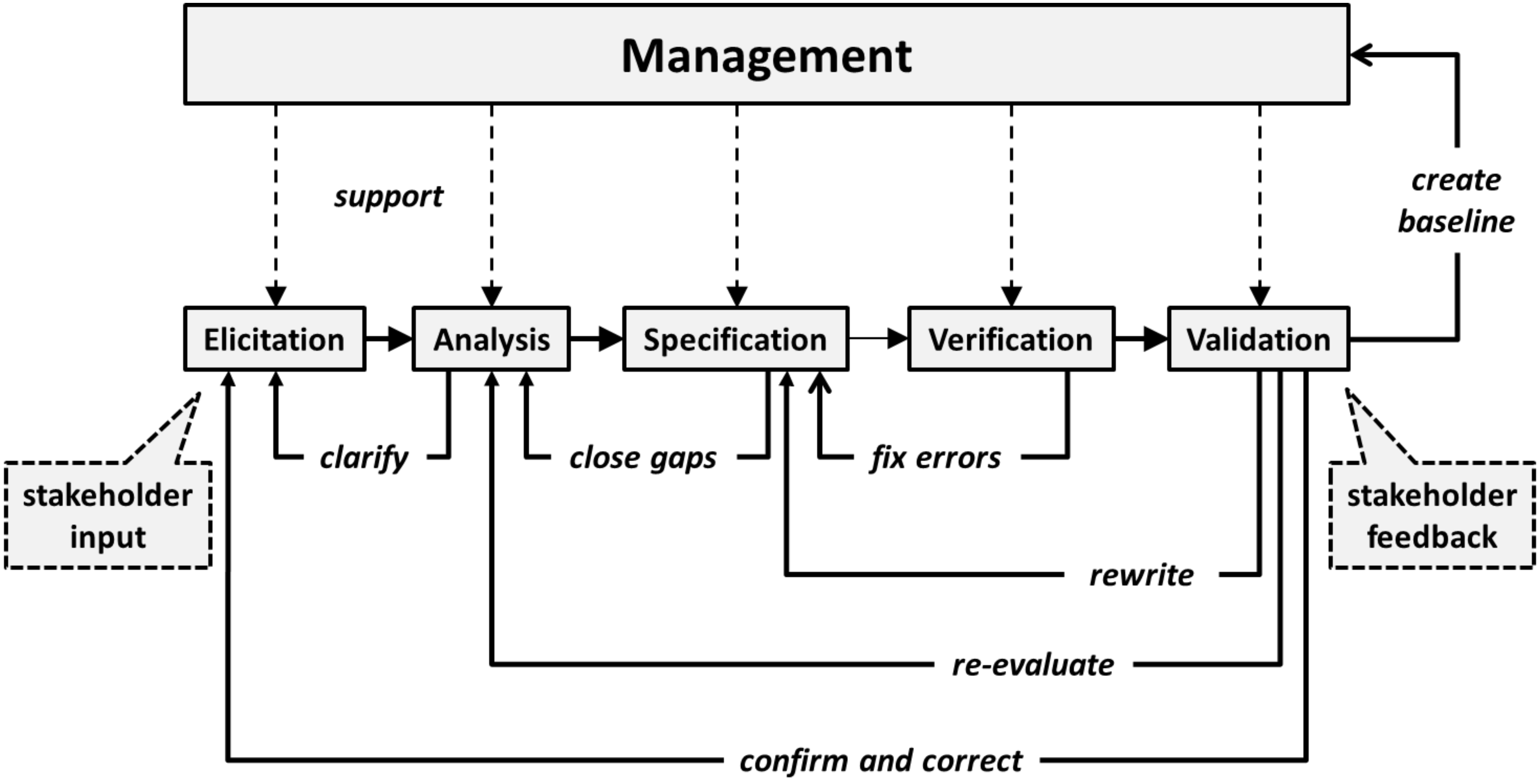


3.2760
software engineering
1. the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software. *ISO/IEC 2382-1:1993, Information technology — Vocabulary — Part 1: Fundamental terms.*01.04.07. **2.** the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software

(IEEE 24765-2010 Systems and Software Engineering – Vocabulary).

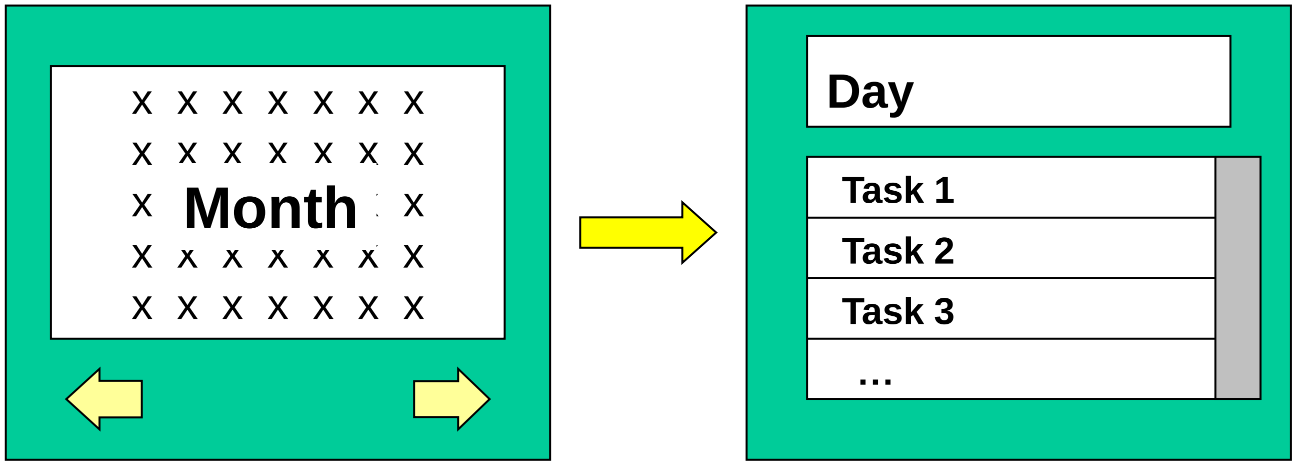
- Requirements Engineering:
  - “Requirements Engineering deals with activities which attempt to understand the exact needs of the users of the software system to be developed and to translate such needs into precise and unambiguous statements which will subsequently be used in the development of the system. ” (Loucopoulos & Karakostas. Systems Requirements Engineering. McGraw-Hill, 1995).
- Special features of software/computer engineering
  - The product is “software”
  - Too much development but not so much engineering
  - Intrinsic need: describe and document what is going to be produced
  - Frequent changes

Requirements Engineering Processes



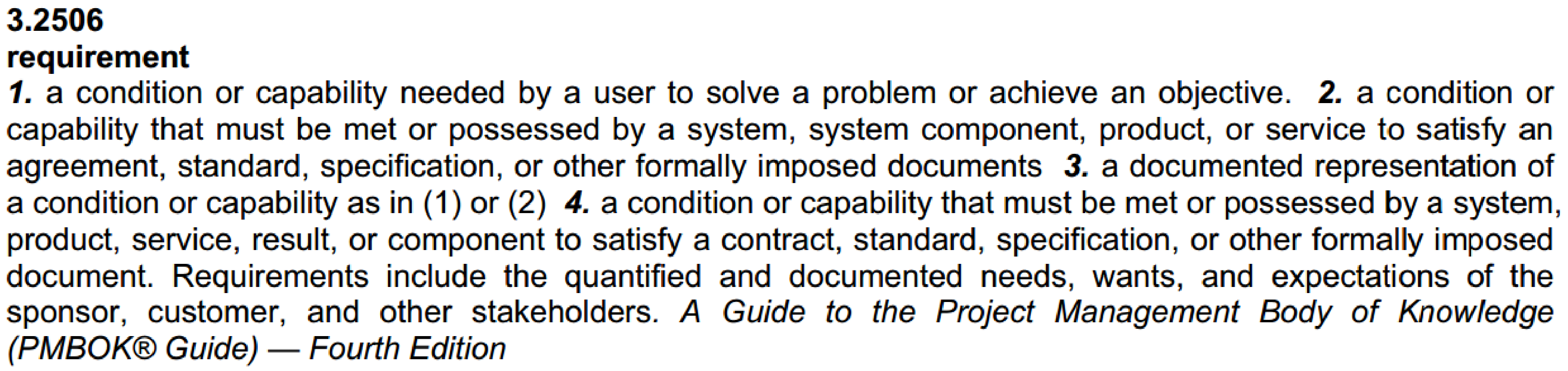
An example…

- “I need something to organize my tasks, an agenda to plan my activities, timetable, etc.”



Definition of “Requirement”

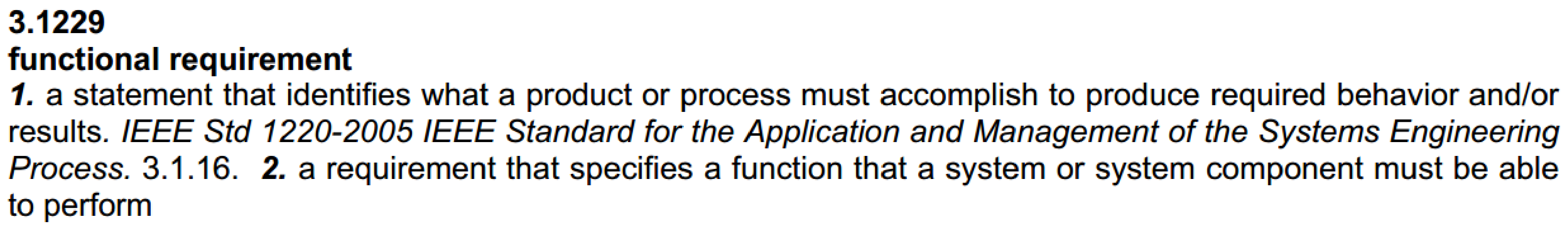
- Following the standard IEEE (24765-2010), a requirement is:



3.2506
requirement
1. a condition or capability needed by a user to solve a problem or achieve an objective. ***2.*** a condition or capability that must be met or possessed by a system, system component, product, or service to satisfy an agreement, standard, specification, or other formally imposed documents ***3.*** a documented representation of a condition or capability as in (1) or (2) ***4.*** a condition or capability that must be met or possessed by a system, product, service, result, or component to satisfy a contract, standard, specification, or other formally imposed document. Requirements include the quantified and documented needs, wants, and expectations of the sponsor, customer, and other stakeholders. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) — Fourth Edition*

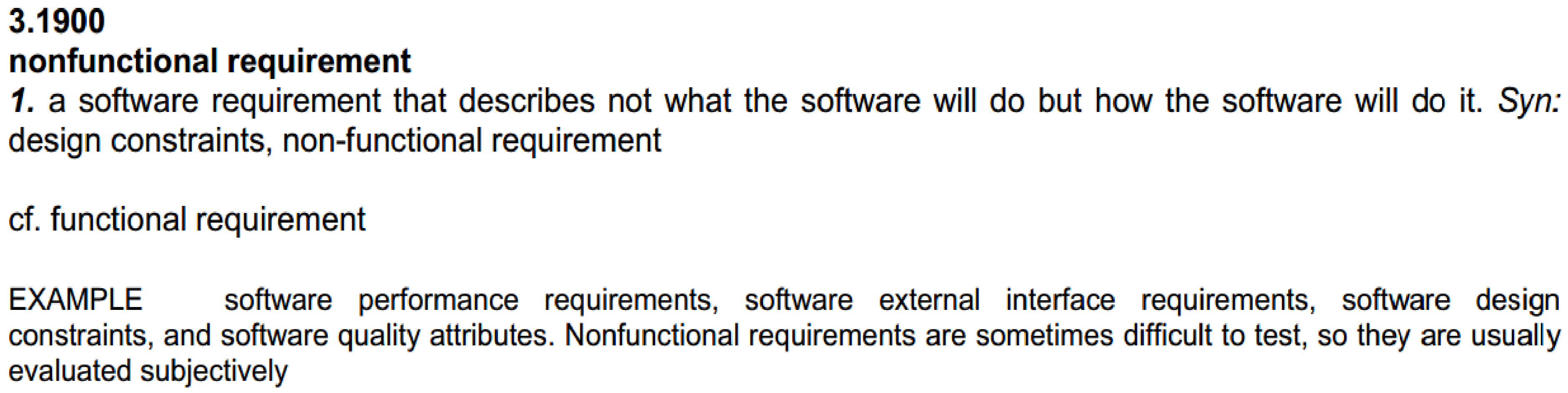
Types of Requirements

Functional Requirements (capabilities)



3.1229
functional requirement
1. a statement that identifies what a product or process must accomplish to produce required behavior and/or results. *IEEE Std 1220-2005 IEEE Standard for the Application and Management of the Systems Engineering Process.* 3.1.16. ***2.*** a requirement that specifies a function that a system or system component must be able to perform

Non-Functional Requirements (restrictions)

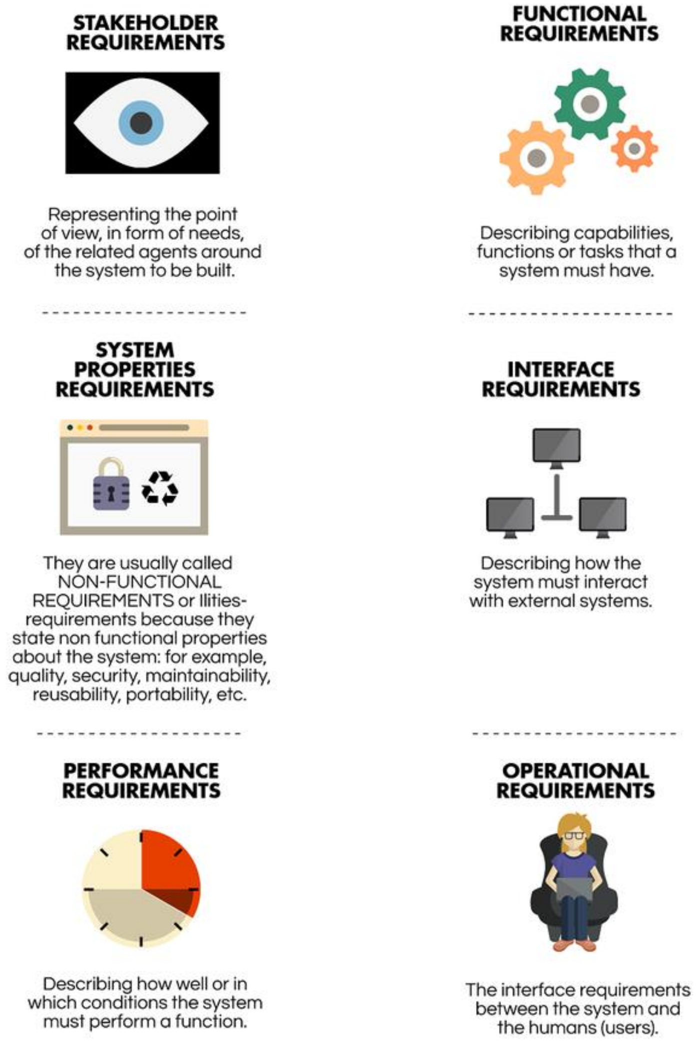


3.1900
nonfunctional requirement
1. a software requirement that describes not what the software will do but how the software will do it. *Syn:* design constraints, non-functional requirement

cf. functional requirement

EXAMPLE software performance requirements, software external interface requirements, software design constraints, and software quality attributes. Nonfunctional requirements are sometimes difficult to test, so they are usually evaluated subjectively

More types of requirements…



STAKEHOLDER REQUIREMENTS

Representing the point of view, in form of needs, of the related agents around the system to be built.

FUNCTIONAL REQUIREMENTS

Describing capabilities, functions or tasks that a system must have.

SYSTEM PROPERTIES REQUIREMENTS

They are usually called NON-FUNCTIONAL REQUIREMENTS or Ilities-requirements because they state non functional properties about the system: for example, quality, security, maintainability, reusability, portability, etc.

INTERFACE REQUIREMENTS

Describing how the system must interact with external systems.

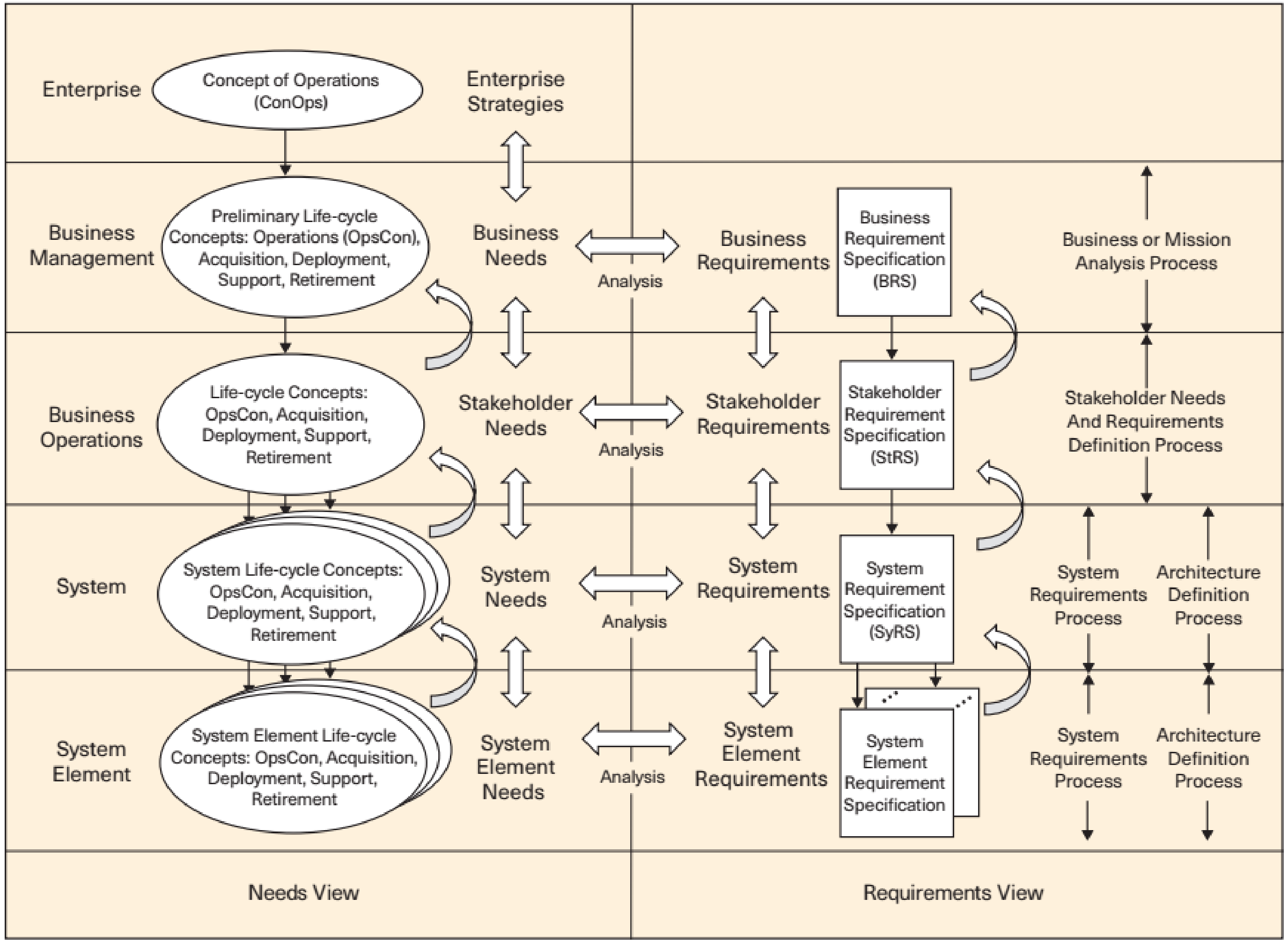
PERFORMANCE REQUIREMENTS

Describing how well or in which conditions the system must perform a function.

OPERATIONAL REQUIREMENTS

The interface requirements between the system and the humans (users).

Transformation of needs into Requirements



Enterprise
Concept of Operations (ConOps)
Enterprise Strategies
Business Management
Preliminary Life-cycle Concepts: Operations (OpsCon), Acquisition, Deployment, Support, Retirement
Business Needs
Analysis
Business Requirements
Business Requirement Specification (BRS)
Business or Mission Analysis Process
Business Operations
Life-cycle Concepts: OpsCon, Acquisition, Deployment, Support, Retirement
Stakeholder Needs
Analysis
Stakeholder Requirements
Stakeholder Requirement Specification (StRS)
Stakeholder Needs And Requirements Definition Process
System
System Life-cycle Concepts: OpsCon, Acquisition, Deployment, Support, Retirement
System Needs
Analysis
System Requirements
System Requirement Specification (SyRS)
System Requirements Process
Architecture Definition Process
System Element
System Element Life-cycle Concepts: OpsCon, Acquisition, Deployment, Support, Retirement
System Element Needs
Analysis
System Element Requirements
System Element Requirement Specification
System Requirements Process
Architecture Definition Process
Needs View
Requirements View

Figure 1. Transformation of needs into requirements (Ryan, 2013).

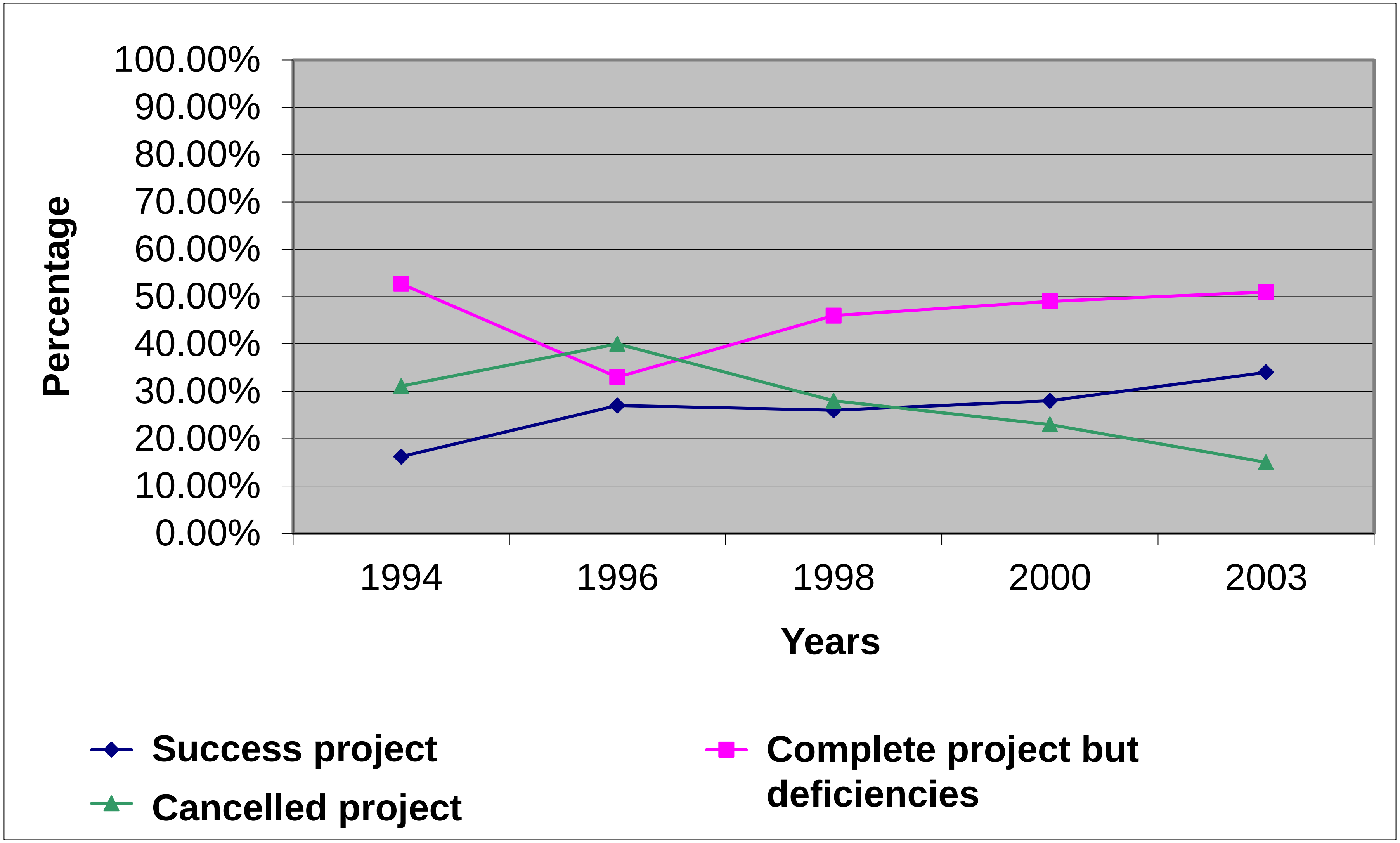
The Chaos Report

- The Standish Group International: technical reports since 1994
- 2003: data about13.522 projects in the field of ICT

	1994	1996	1998	2000	2003
Success project	16%	27%	26%	28%	34%
Complete project but deficiencies	53%	33%	46%	49%	51%
Cancelled project	31%	40%	28%	23%	15%

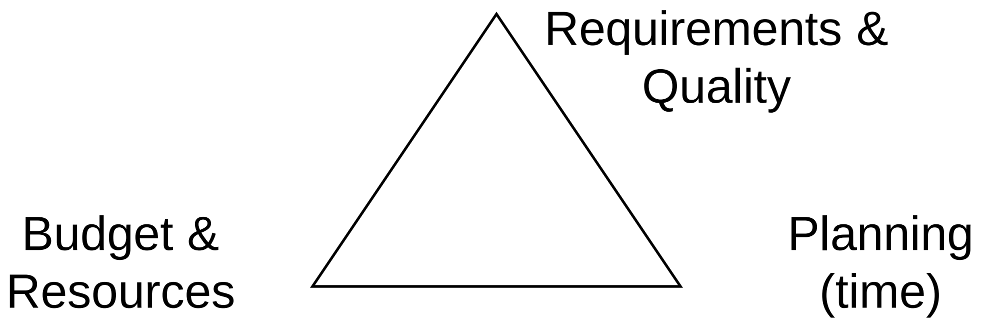
E.g. https://net.educause.edu/ir/library/pdf/NCP08083B.pdf

The Chaos Report

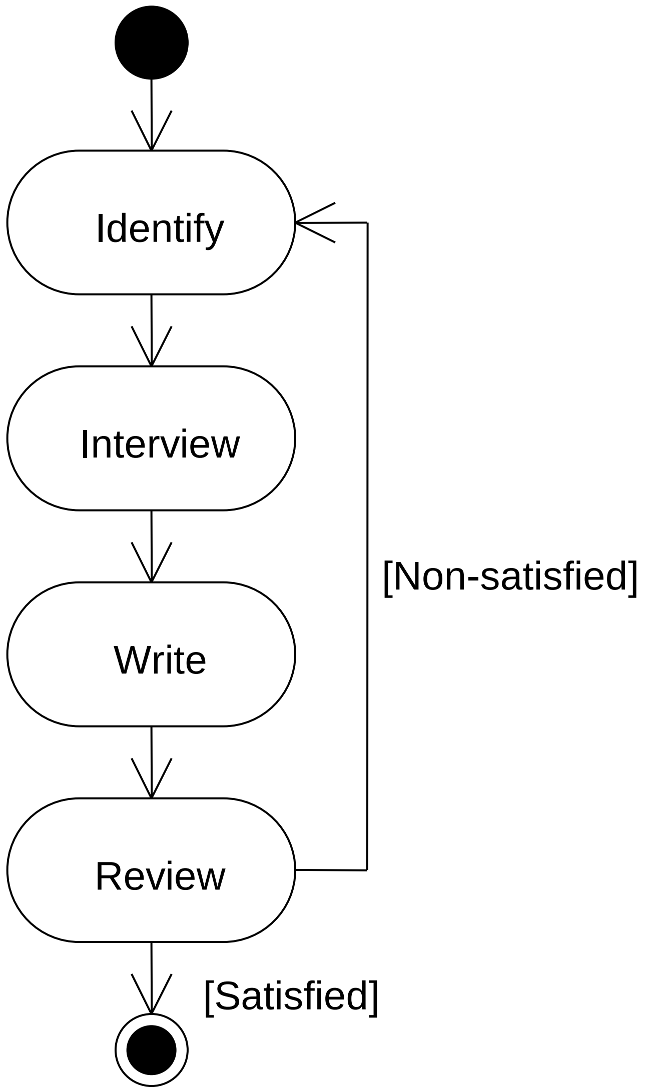


Elicitation and description of requirements

- Sources of requirements
- Work plan to gather requirements
- Identification of *stakeholders*
  - Who have an interest in the product?
  - Who is the client?
  - Conflicting interests, contradictory requirements
  - Negotiate a balance between:



- How to make an interview…
  - Before, After...



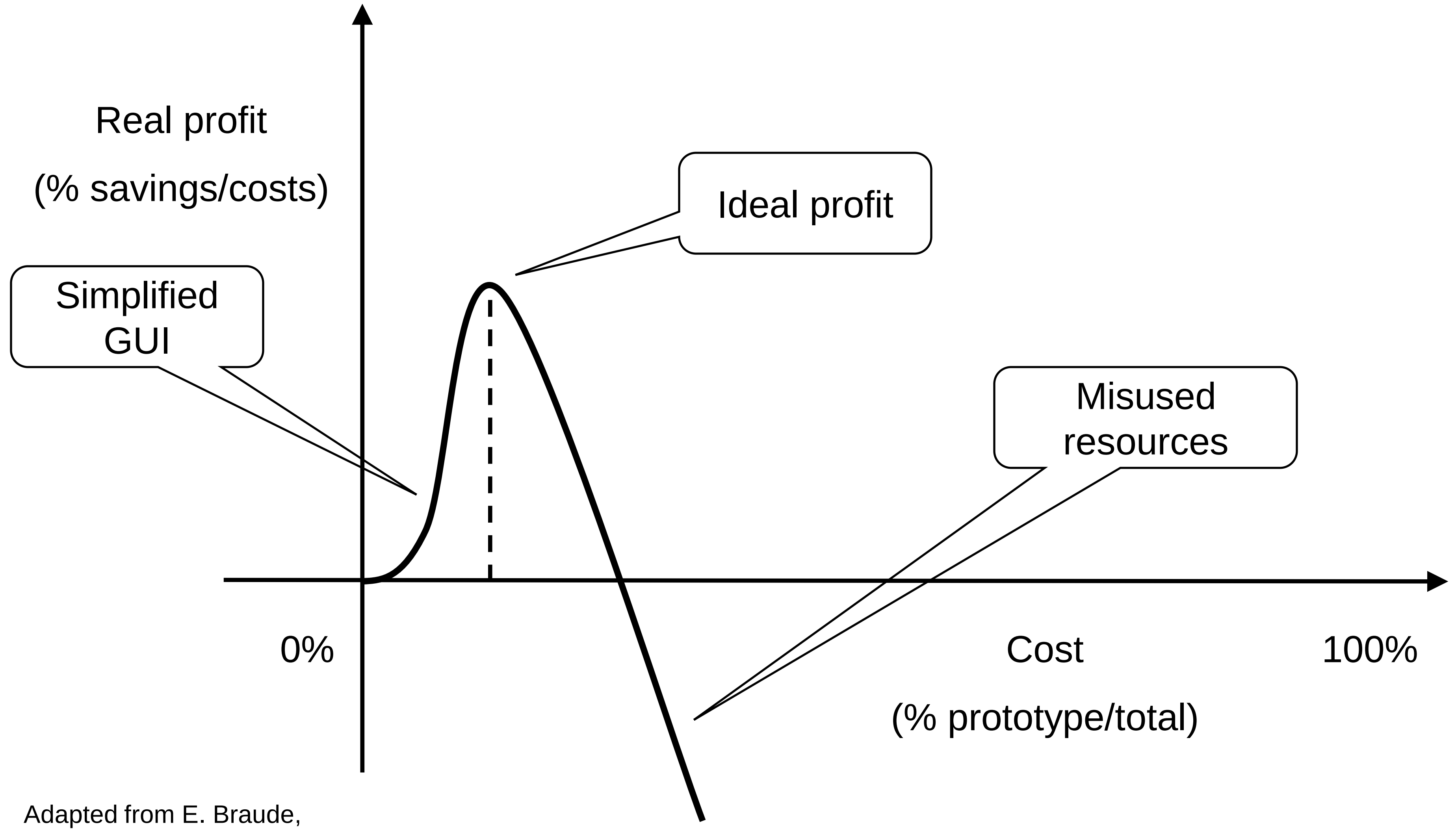
How to elicitate and describe requirements: techniques

- Text-based techniques
  - Relatively easy to deal with clients without any specific background
    - Raw text in human language
    - Structured text and technical language
    - Use cases
    - Table for mapping user roles and functionalities
- Graphical techniques
  - Require a kind of background to interact with the client
  - Warning: Potentially we can design instead of analyzing
    - Data Flow diagrams
    - Activity diagrams
    - State diagrams
- Mock user interfaces and prototypes
  - Just for analysis (it is NOT design) and it requires an analysis of the costs

Auction platform

Role	Service(s)
Anonymous	Get list of articles Register as a seller Register as a client
Seller	Announce a new article for auction Modify articles
Client	Bid for an article
Seller Client Administrator	Formalize the process
Administrator	...

Return of investment when using prototypes



Adapted from E. Braude,
Software Engineering: An Object-Oriented Perspective

A good Software Requirements Specification (SRS)…

- **Correct**
- **Unambiguous**
- **Complete**
- **Consistent**
- **Ranked for importance and/or stability**
- **Verifiable**
- **Modifiable**
- **Traceable**

See the next definitions from the standard IEEE 830, 1998

Correct

- An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet.
- There is no tool or procedure that ensures correctness.
- The SRS should be compared with any applicable superior specification, such as a system requirements specification, with other project documentation, and with other applicable standards, to ensure that it agrees.
- Alternatively the customer or user can determine if the SRS correctly reflects the actual needs.
- Traceability makes this procedure easier and less prone to error.

Unambiguous

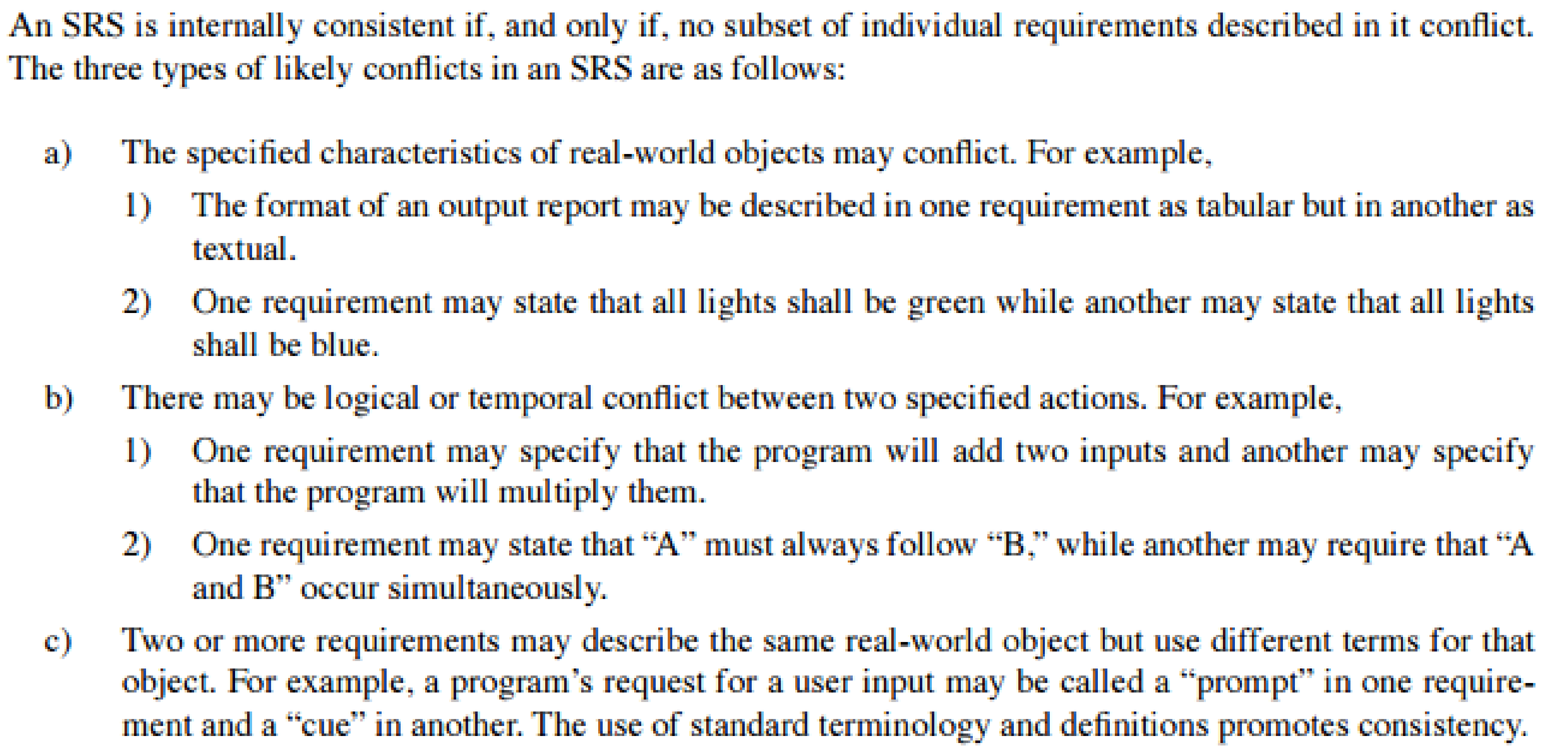
- An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the final product be described using a single unique term.
- In cases where a term used in a particular context could have multiple meanings, the term should be included in a glossary where its meaning is made more specific.
- An SRS is an important part of the requirements process of the software life cycle and is used in design, implementation, project monitoring, verification and validation, and in training.
- The SRS should be unambiguous both to those who create it and to those who use it. However, these groups often do not have the same background and therefore do not tend to describe software requirements the same way. Representations that improve the requirements specification for the developer may be counterproductive in that they diminish understanding to the user and vice versa.

Complete

- An SRS is complete if, and only if, it includes the following elements:
  - All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specification should be acknowledged and treated.
  - Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values.
  - Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure

Consistent

- Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct.



An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict. The three types of likely conflicts in an SRS are as follows:

a) The specified characteristics of real-world objects may conflict. For example,
   1) The format of an output report may be described in one requirement as tabular but in another as textual.
   2) One requirement may state that all lights shall be green while another may state that all lights shall be blue.

b) There may be logical or temporal conflict between two specified actions. For example,
   1) One requirement may specify that the program will add two inputs and another may specify that the program will multiply them.
   2) One requirement may state that "A" must always follow "B," while another may require that "A and B" occur simultaneously.

c) Two or more requirements may describe the same real-world object but use different terms for that object. For example, a program's request for a user input may be called a "prompt" in one requirement and a "cue" in another. The use of standard terminology and definitions promotes consistency.

Verifiable

- An SRS is verifiable if, and only if, every requirement stated therein is verifiable.
- A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement. In general any ambiguous requirement is not verifiable.
- Non-verifiable requirements include statements such as “works well,” “good human interface,” and “shall usually happen.”
- These requirements cannot be verified because it is impossible to define the terms “good,” “well,” or “usually.”
- The statement that “the program shall never enter an infinite loop” is non-verifiable because the testing of this quality is theoretically impossible.

Modifiable

- An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style. Modifiability generally requires an SRS to
  - a) Have a coherent and easy-to-use organization with a table of contents, an index, and explicit crossreferencing;
  - b) Not be redundant (i.e., the same requirement should not appear in more than one place in the SRS);
  - c) Express each requirement separately, rather than intermixed with other requirements.
- Redundancy itself is not an error, but it can easily lead to errors. Redundancy can occasionally help to make an SRS more readable, but a problem can arise when the redundant document is updated.
- For instance, a requirement may be altered in only one of the places where it appears. The SRS then becomes inconsistent. Whenever redundancy is necessary, the SRS should include explicit cross-references to make it modifiable

Traceable

- An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation.
- The following two types of traceability are recommended:
  - a) Backward traceability (i.e., to previous stages of development). This depends upon each requirement explicitly referencing its source in earlier documents.
  - b) Forward traceability (i.e., to all documents spawned by the SRS). This depends upon each requirement in the SRS having a unique name or reference number.
- The forward traceability of the SRS is especially important when the software product enters the operation and maintenance phase.
- As code and design documents are modified, it is essential to be able to ascertain the complete set of requirements that may be affected by those modifications.

The seven sins of the specifier…

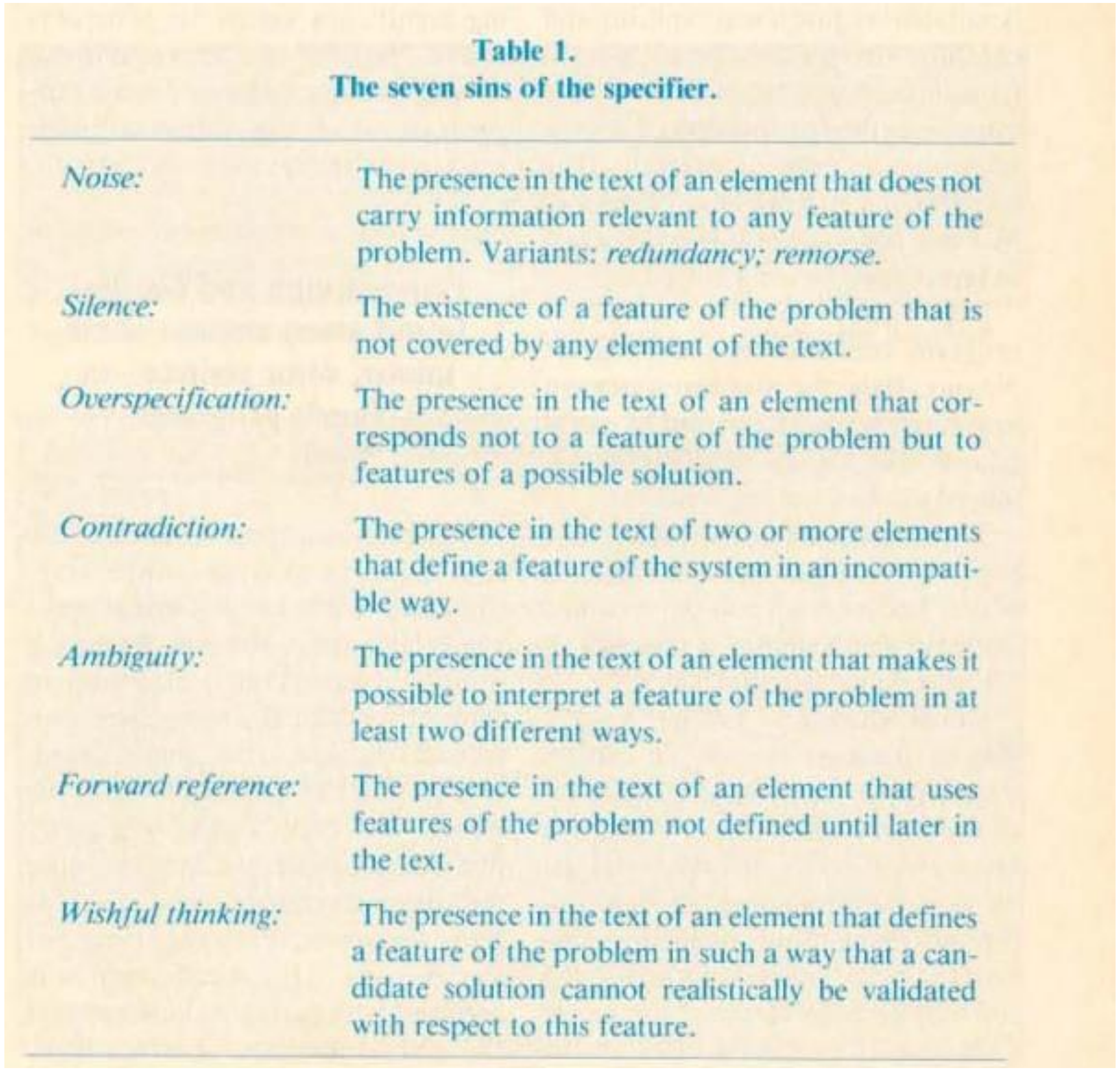


Table 1.
The seven sins of the specifier.

Noise:	The presence in the text of an element that does not carry information relevant to any feature of the problem. Variants: *redundancy; remorse.*
Silence:	The existence of a feature of the problem that is not covered by any element of the text.
Overspecification:	The presence in the text of an element that corresponds not to a feature of the problem but to features of a possible solution.
Contradiction:	The presence in the text of two or more elements that define a feature of the system in an incompatible way.
Ambiguity:	The presence in the text of an element that makes it possible to interpret a feature of the problem in at least two different ways.
Forward reference:	The presence in the text of an element that uses features of the problem not defined until later in the text.
Wishful thinking:	The presence in the text of an element that defines a feature of the problem in such a way that a candidate solution cannot realistically be validated with respect to this feature.

- *Bertrand Meyer, On Formalism in Specification, IEEE Software, Jan 1985, pp. 6-26*
- *Source: http://se.ethz.ch/~meyer/publications/ieee/formalism.pdf*

SMART Requirements

	Concept	Description
S	**S**pecific	"A good requirement is specific and not generic. It should not be open to mis-interpretation when read by others. This is the most important attribute to get correct."
M	**M**easurable	"This answers whether you will be able to verify the completion of the project. You should avoid signing up for any requirement that cannot be verified as complete. These are especially risky when you use non-quantitative terms (best, optimal, fastest) for acceptance criteria.."
A	**A**chievable	"The requirement is physically able to be achieved given existing circumstances"
R	**R**ealistic	"Whether the requirement is realistic to deliver when considering other constraints of the project and requirements."
T	**T**ime-bound	"Each requirement should be time-bound or specify by *when* or *how fast* a requirement needs to be completed or executed. "

See discussion at https://jessica80304.wordpress.com/2008/08/04/smart-requirements/

SMART Requirements

"I believe that this nation should commit itself to achieving the goal, before this decade is out, of landing a man on the Moon and returning him safely to Earth"

Structure of a requirements specification

- A mid-size project can contain hundreds of requirements
- It is necessary to write requirement to no forget nothing:
  - They can be used as part of an agreement
  - They are the input for the next stage: design
  - The will serve us to verified the produced software
- A proper organization is a cornerstone
- Key points:
  - Use of standards that have been defined to organize requirements
  - Have a clear overall goal that the system must meet
  - Use textual and visual descriptions
  - Sort and group requirements according to a “sound and logic” criteria
  - Create links between requirements to have a better understanding of the interactions and relationships
  - Create links between requirements and other assets

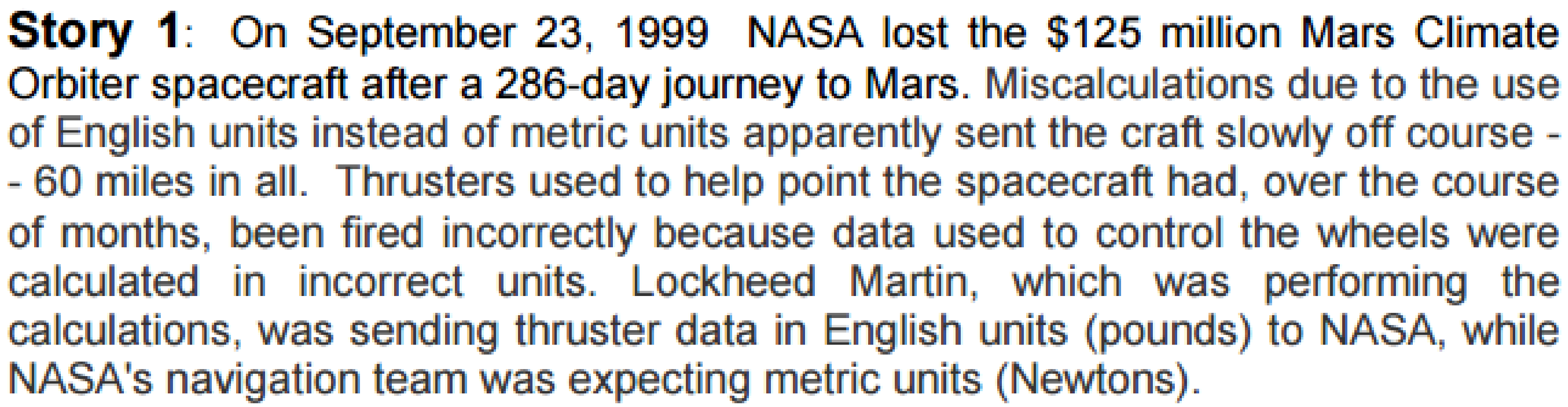
Completeness:
How can we ensure that a requirements specification is complete?

- How to not forget nothing?
  - Peer-review: colleagues, senior analysts, experts in the topics, client(s) and any other stakeholder
  - Use of check-lists
  - Compare the specification against existing taxonomies or knowledge in the same domain
  - Reuse of previous requirements (in similar projects)
    - Coarse-grained reuse: reusable components
    - Fine-grained reuse: an advanced search mechanism is required

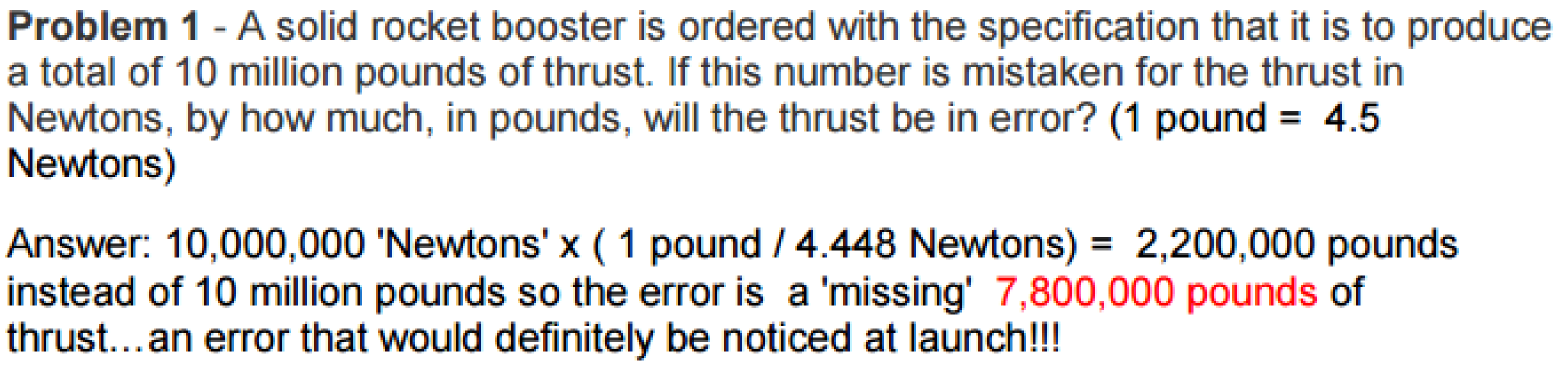
Consistency: How can we ensure that a requirements specification is consistent?

- The first step towards consistency is to avoid redundancy…
  - … inconsistencies imply re-work
- Group-based reviews allow us to detect redundancy
- Automatic techniques:
  - Semantic graph comparison
  - Detection of inconsistent units
    - “Metric mishap caused loss of NASA orbiter”→125 million $
      - http://spacemath.gsfc.nasa.gov/weekly/6Page53.pdf
    - Global reviews of the specification
    - Automatic detection of inconsistent units

Example 1 of an inconsistency problem



Story 1: On September 23, 1999 NASA lost the $125 million Mars Climate Orbiter spacecraft after a 286-day journey to Mars. Miscalculations due to the use of English units instead of metric units apparently sent the craft slowly off course -- 60 miles in all. Thrusters used to help point the spacecraft had, over the course of months, been fired incorrectly because data used to control the wheels were calculated in incorrect units. Lockheed Martin, which was performing the calculations, was sending thruster data in English units (pounds) to NASA, while NASA's navigation team was expecting metric units (Newtons).



Problem 1 - A solid rocket booster is ordered with the specification that it is to produce a total of 10 million pounds of thrust. If this number is mistaken for the thrust in Newtons, by how much, in pounds, will the thrust be in error? (1 pound = 4.5 Newtons)

Answer: 10,000,000 'Newtons' x (1 pound / 4.448 Newtons) = 2,200,000 pounds instead of 10 million pounds so the error is a 'missing' 7,800,000 pounds of thrust…an error that would definitely be noticed at launch!!!

Example 2 of an inconsistency problem

Story 2: On January 26, 2004 at Tokyo Disneyland's Space Mountain, an axle broke on a roller coaster train mid-ride, causing it to derail. The cause was a part being the wrong size due to a conversion of the master plans in 1995 from English units to Metric units. In 2002, new axles were mistakenly ordered using the pre-1995 English specifications instead of the current Metric specifications.

Problem 2 - A bolt is ordered with a thread diameter of 1.25 inches .What is this diameter in millimeters? If the order was mistaken for 1.25 centimeters, by how many millimeters would the bolt be in error? Answer: 1- inch = 25.4 millimeters so 1.25 inches x (25.4 mm/ 1 inch) = 31.75 millimeters. Since 1.25 centimeters = 12.5 millimeters, the bolt would delivered 31.75 - 12.5 = 19.25 millimeters too small!

Consistent Specifications

- Semantic graph comparison:

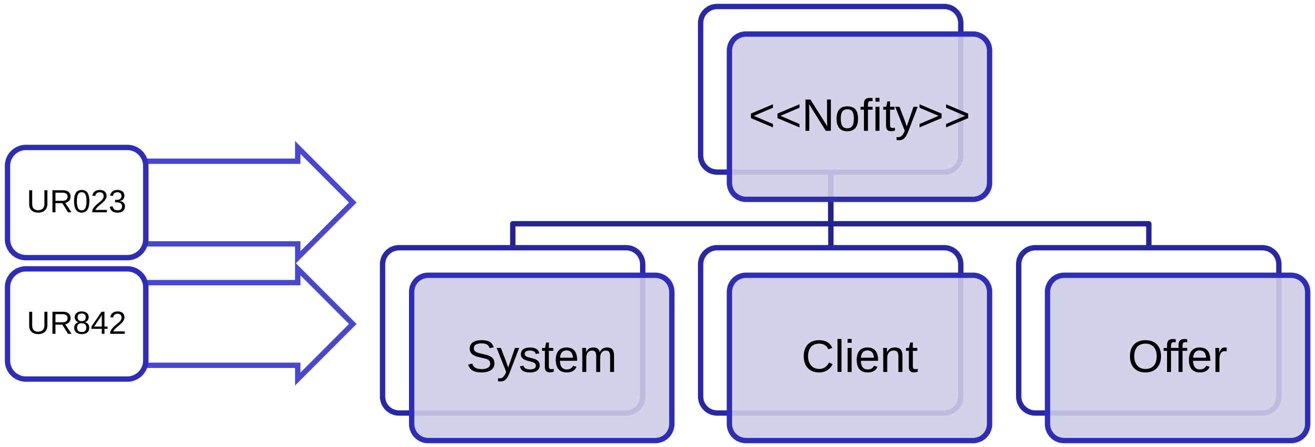
  UR001: ….

  UR023: The system shall send weekly notifications about new offers to all clients.

  URxxx: …

  UR842: The application shall be able to periodically notify clients about new offers.

  UR999: …



A requirements specification is NOT a novel…

Novel	Requirements Specification
Multiple storytelling styles	Plain storytelling style
Many Rhythm and Meter	• Text shall be simple and plain to ease readers the reading and understanding • Text shall follow a predefined set of restricted grammars • Use of active voice • Avoid technical terms that have not been defined such as acronyms. • Use of an agreed controlled vocabulary
Imagination	We should leave no stone unturned. Requirements shall be SMART

A requirements specification is NOT a novel…(2)

Novel	Requirements Specification
It is possible to make references to other texts or implicit messages.	No assuming any previous knowledge: • Self-contained. • It is possible to include glossaries
Mix of different thematic threads, go and back in time, etc.	Need of a logical and structured order • Requirements shall be atomic.
They are written with enriched text-processors.	We have to use specific tools to have features such as • Unique identifiers • Atomicity • Organization and traceability mechanisms • Reuse (individual and general) • Quality checking (individual and general)

How to write good requirements: quality indicators

- Size:
  - The proper one, no too long no too short
  - The metrics are: number of characters, words and paragraphs
- Readability:
  - As much as possible
  - Some text processors such as MS Word offers this type of checking
- Form
  - Use of Imperative instead of conditional. Unify how you write your requirements
    - The user shall (open, send, register, etc.): ok
    - The user shall be able to (open, send, register): a bit overloaded
- Voice:
  - Active voice instead of passive voice
- Optional or speculative sentences:
  - Avoid to write expressions such as “perhaps”, “maybe”, “usually”, “most of times”, “probably”, etc.
- Ambiguous expressions (adverbs and adjectives)
  - Avoid to write expression such as “quickly”, “friendly”, etc.

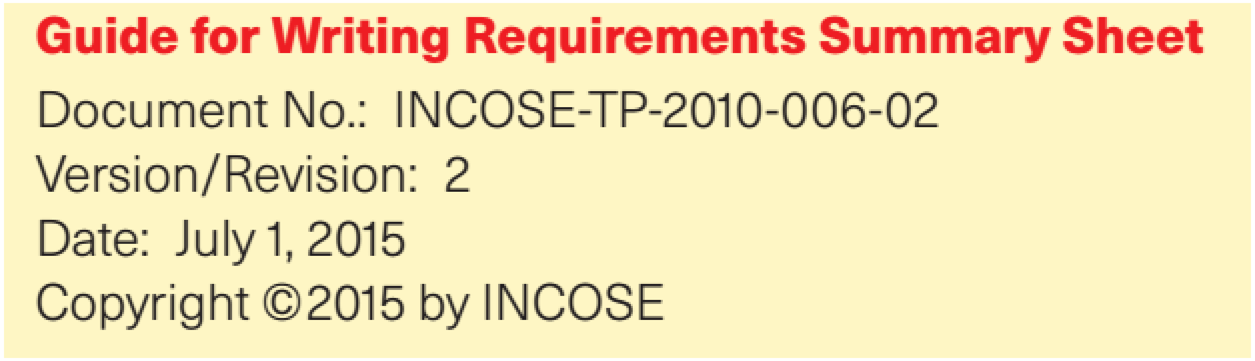
Requirements quality indicators

- Subjective expressions:
  - Avoid opinion-based sentences “I believe…”, “In my opinion…”
- Use of pronouns:
  - Avoid the use of pronouns (implicit subjects)
  - The reader can hesitate to which noun the pronoun refers to
- Connectors
  - Avoid the use of too many connectors since it can imply that the requirement is defining more than one functionality
  - Too much details
- Negation
  - More than one negative work can be confusing: more than one interpretation
- Incomplete sentences
  - Avoid “…”, “etc.”
- Design and workflow
  - Avoid terms that can have a meaning in the design phase such as “foreign key”, “hash table”, etc.

Requirements quality indicators

- Number of domain terms:
  - Too many domain terms in just one requirement can imply that the requirement is defining more than one need
  - Too much detail
- Number of domain verbs:
  - Too many domain verbs in just one requirement can imply that the requirement is defining more than one functionality
  - Too much detail
- Acronyms and abbreviations
  - Only allowed if they are defined in other section such as a glossary
- Grammar structure (pattern):

User	The Call center operator
Action	… shall be able to see…
Object	… the details of a client…
Quantifier	… in less than 2 seconds.



Guide for Writing Requirements Summary Sheet

Document No.: INCOSE-TP-2010-006-02

Version/Revision: 2

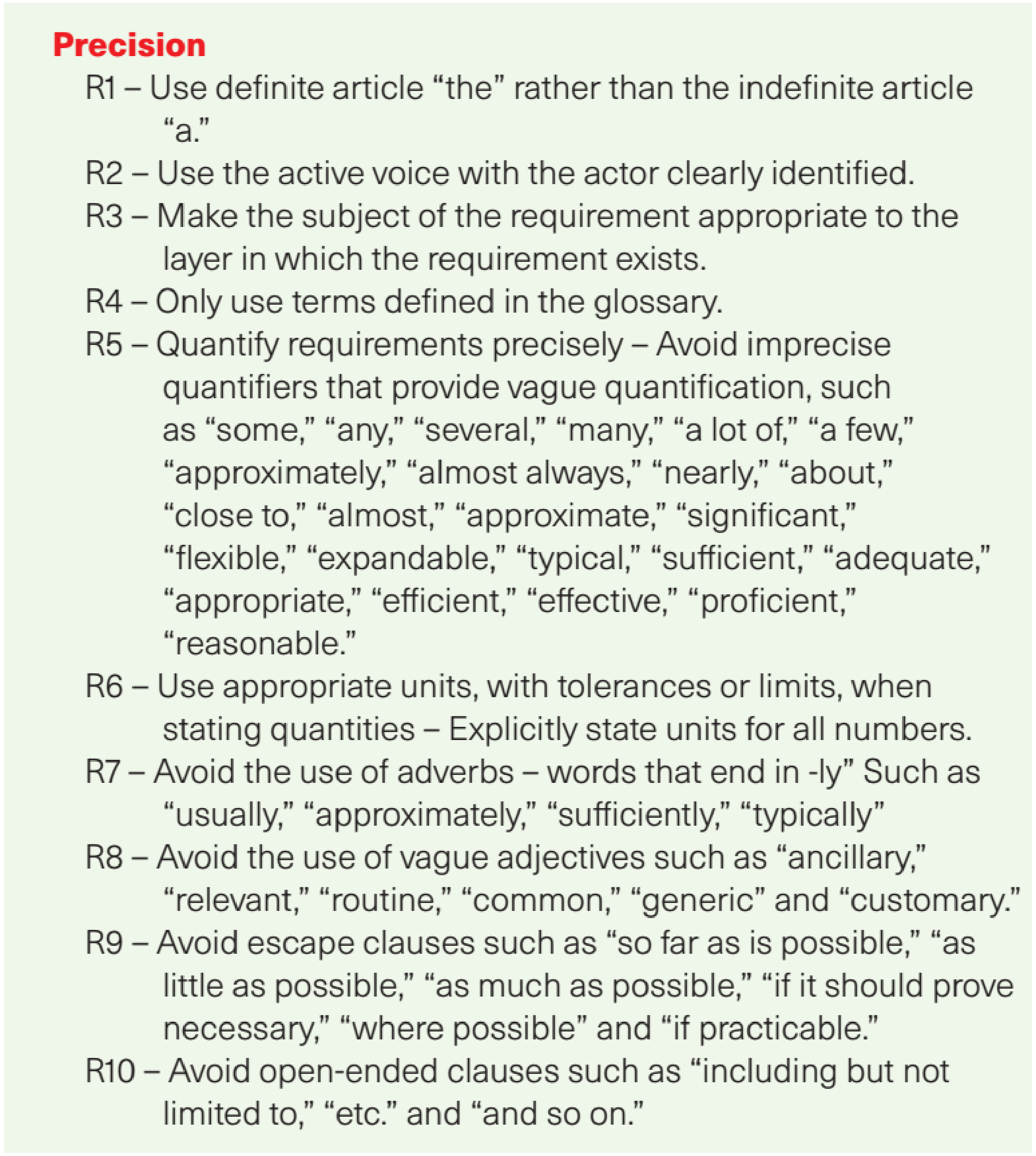
Date: July 1, 2015

Copyright ©2015 by INCOSE

Rules for Requirement Statements and Sets of Requirements

GUIDE FOR WRITING REQUIREMENTS, INCOSE, 2015

JUST FOR YOUR INFORMATION! (SLIDES 40-46)
(NOT TO STUDY)



Precision

R1 – Use definite article "the" rather than the indefinite article "a."

R2 – Use the active voice with the actor clearly identified.

R3 – Make the subject of the requirement appropriate to the layer in which the requirement exists.

R4 – Only use terms defined in the glossary.

R5 – Quantify requirements precisely – Avoid imprecise quantifiers that provide vague quantification, such as "some," "any," "several," "many," "a lot of," "a few," "approximately," "almost always," "nearly," "about," "close to," "almost," "approximate," "significant," "flexible," "expandable," "typical," "sufficient," "adequate," "appropriate," "efficient," "effective," "proficient," "reasonable."

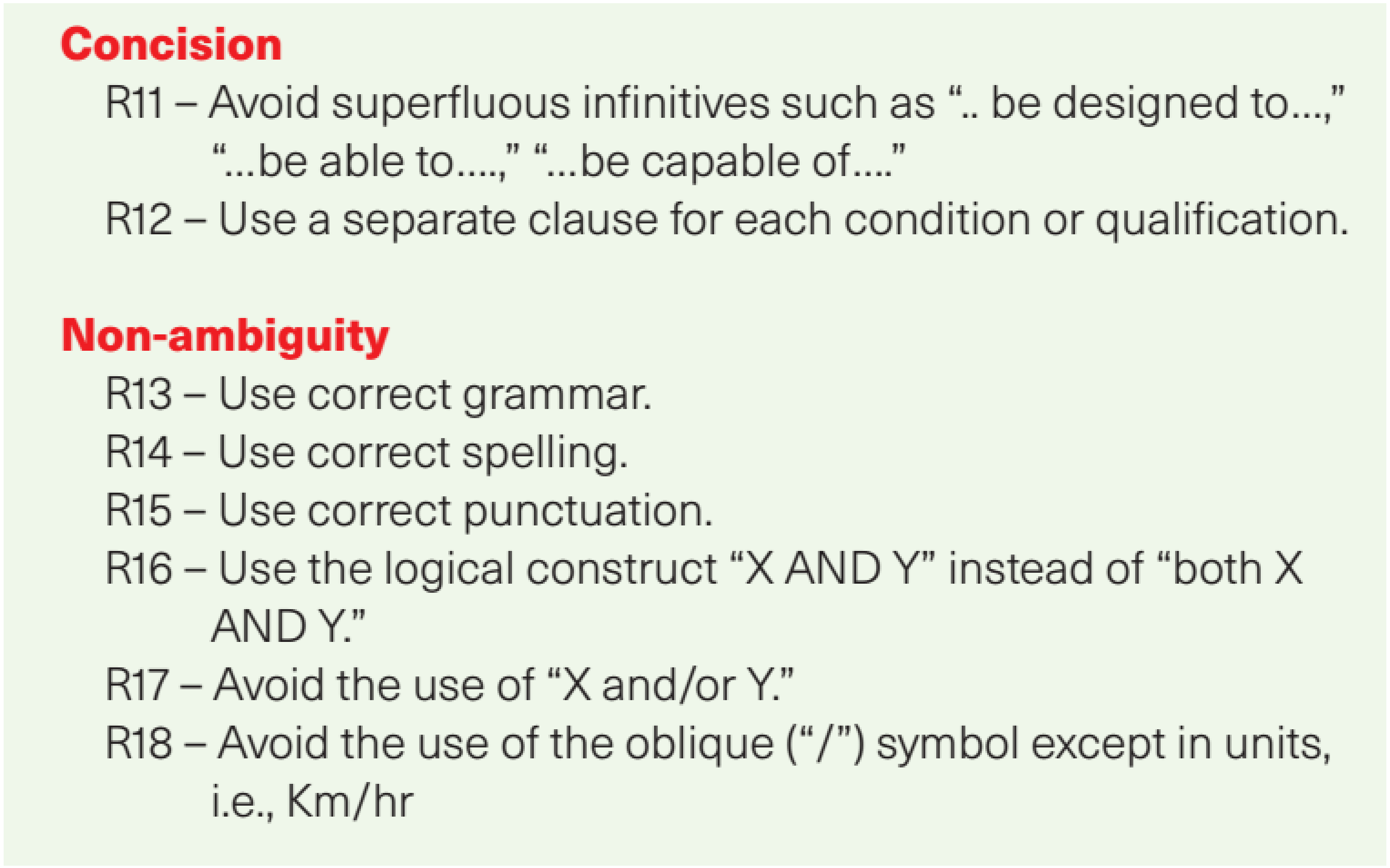
R6 – Use appropriate units, with tolerances or limits, when stating quantities – Explicitly state units for all numbers.

R7 – Avoid the use of adverbs – words that end in -ly" Such as "usually," "approximately," "sufficiently," "typically"

R8 – Avoid the use of vague adjectives such as "ancillary," "relevant," "routine," "common," "generic" and "customary."

R9 – Avoid escape clauses such as "so far as is possible," "as little as possible," "as much as possible," "if it should prove necessary," "where possible" and "if practicable."

R10 – Avoid open-ended clauses such as "including but not limited to," "etc." and "and so on."



Concision

R11 – Avoid superfluous infinitives such as ".. be designed to...," "...be able to....," "...be capable of...."

R12 – Use a separate clause for each condition or qualification.

Non-ambiguity

R13 – Use correct grammar.

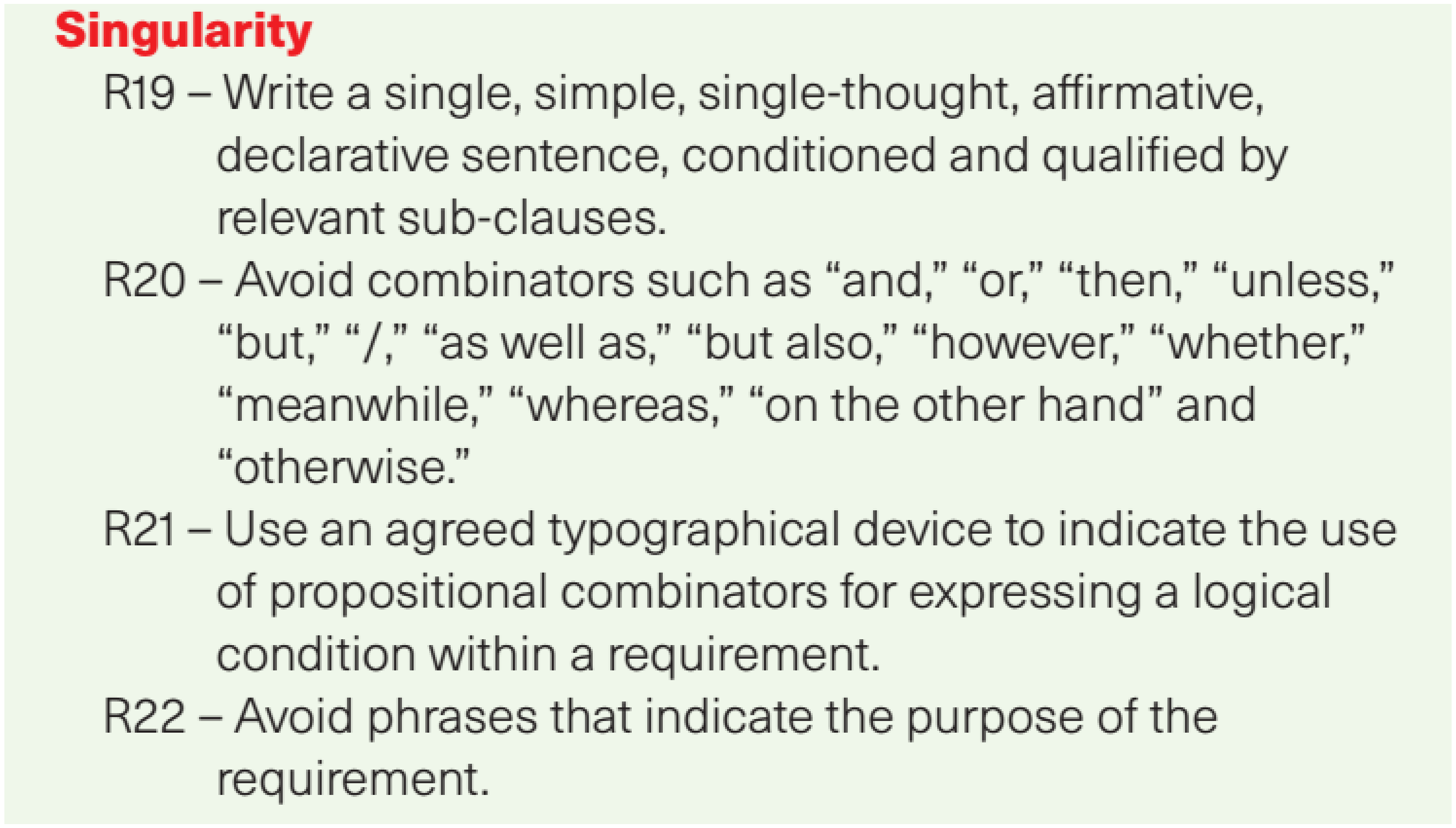
R14 – Use correct spelling.

R15 – Use correct punctuation.

R16 – Use the logical construct "X AND Y" instead of "both X AND Y."

R17 – Avoid the use of "X and/or Y."

R18 – Avoid the use of the oblique ("/") symbol except in units, i.e., Km/hr



Singularity

R19 – Write a single, simple, single-thought, affirmative, declarative sentence, conditioned and qualified by relevant sub-clauses.

R20 – Avoid combinators such as “and,” “or,” “then,” “unless,” “but,” “/,” “as well as,” “but also,” “however,” “whether,” “meanwhile,” “whereas,” “on the other hand” and “otherwise.”

R21 – Use an agreed typographical device to indicate the use of propositional combinators for expressing a logical condition within a requirement.

R22 – Avoid phrases that indicate the purpose of the requirement.

R23 – Avoid parentheses and brackets containing subordinate text.

R24 – Enumerate sets of entities as explicit requirements instead of using generalizations.

R25 – When a requirement is related to complex behavior, refer to the supporting diagram or model rather than a complex textual description

Completeness

R26 – Avoid the use of pronouns.

R27 – Avoid using headings to support explanation of subordinate requirements.

Realism

R28 – Avoid using unachievable absolutes such as 100% reliability or 100% availability.

Conditions

R29 – State applicability conditions explicitly instead of leaving applicability to be inferred from the context.

R30 – Express the propositional nature of a condition explicitly instead of giving lists of conditions.

Uniqueness

R31 – Classify the requirement according to the aspects of the problem or system it addresses.

R32 – Express each requirement once and only once.

Abstraction

R33 – Avoid stating a solution – focus on the problem "what" rather than the solution "how."

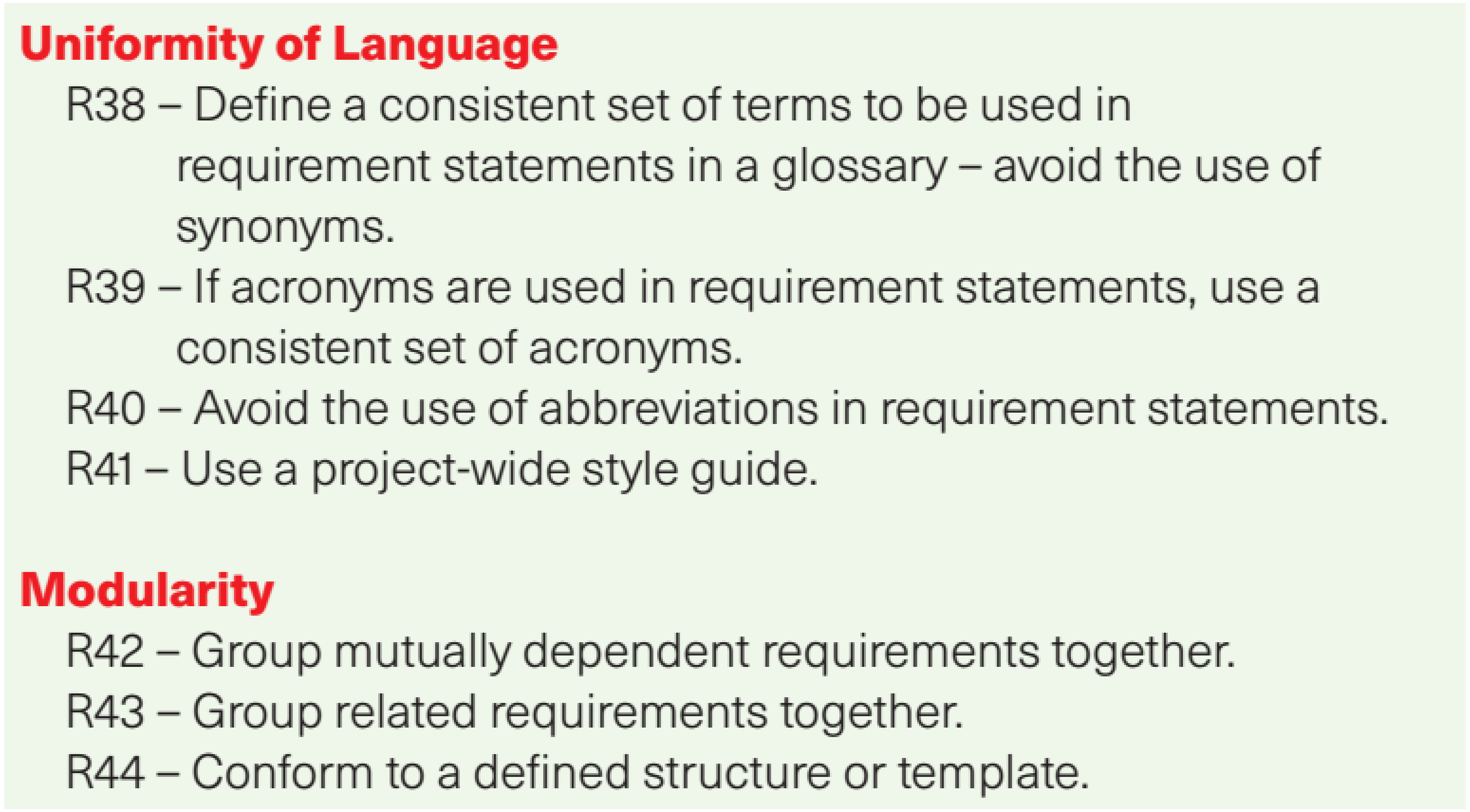
Quantifiers

R34 – Use "each" instead of "all," "any" or "both" when universal quantification is intended

R35 – Define the range of acceptable values associated with quantities.

R36 – Provide specific measurable performance targets.

R37 – Define temporal dependencies explicitly instead of using indefinite temporal keywords.



Uniformity of Language

R38 – Define a consistent set of terms to be used in requirement statements in a glossary – avoid the use of synonyms.

R39 – If acronyms are used in requirement statements, use a consistent set of acronyms.

R40 – Avoid the use of abbreviations in requirement statements.

R41 – Use a project-wide style guide.

Modularity

R42 – Group mutually dependent requirements together.

R43 – Group related requirements together.

R44 – Conform to a defined structure or template.

Examples of Patterns

Pattern
<system> may <action>
<system> may <action> <entity>
<system> may be <state>
<system> shall <action>
<system> shall <action> <entity>
<system> shall allow <entity> to be <state>
<system> shall be <entity>
<system> shall have <entity>
<system> shall have <quality factor> or at least <quantity> <unit>
<system> shall have <quality factor> or at the most <quantity> <unit>
<system> shall not <action>
<system> shall not <action> <entity>
<system> shall not allow <action>
<system> shall not allow <action> <entity>
<system> shall not allow <entity> <action>
<user> shall be able to <action>
<system> shall be <state>
<system> shall be <state> <quantity> <unit>

http://slides.com/josemariaalvarez/patterns_requirements_engineering#/

Common errors (1): conditional clauses

- *“The user information* ***should*** *be stored in a hash table (in the main memory) or in database table having a foreign key to the user table.”*
  - Avoid the use of conditional clauses
  - Use active voice and imperative clauses
- *“The user information* ***shall*** *be stored in a hash table (in the main memory) or in database table having a foreign key to the user table.”*

Common errors (2): design details

- *“The user information shall be stored in a hash table (in the main memory) or in database table having a foreign key to the user table.”*
  - Avoid design details
- *“The user information shall be stored **in the storage subsystem**”*

Common errors (3): optional clauses

- *"The administrator shall be able to create, retrieve, update and delete users' information. Optionally, he has also able to generate a report and send an email to the client."*
  - An optional characteristic shall be expressed as an attribute of the requirement, no as a text within the requirement
- *"The administrator shall be able to create, retrieve, update and delete users' information. He has also able to generate a report and send an email to the client."*
  - Necessity: optional.

Common errors (4): atomicity

- *"The administrator shall be able to create, retrieve, update and delete users' information. He has also able to generate a report and send an email to the client."*
  - Try to write atomic requirements (one per need). Too many domain verbs in just one requirement can imply that the requirement is defining more than one functionality.
- "The administrator shall be able to add new users"
  - Necessity: mandatory.
- …
- "The administrator shall be able to send an email to the client".
  - Necessity : optional.

Common errors (5): acronyms and ambiguity

- "The system shall be able to import ABC files. The process shall be user friendly and fast."
  - Use of acronyms if they are defined in a glossary and can be understood for all stakeholders.
  - Terms such as friendly and fast are hard to measure. Therefore, it is not possible to verify them.
  - Use measurement units (and magnitudes) to define performance (or any other quality characteristic).
  - Use other means (e.g. WAI AA*) to clearly define how "user friendly" the system shall be (in this case in terms of accessibility)

** Web Accesibility Initiative, Nivel Doble-A*

Common errors (6): punctuation and readability

- “The administrator shall be able to generate invoices associated to different companies and he shall be informed of the pending invoices to generate an email to inform them.”
  - The proper use of punctuation is a cornerstone to ease the reading.
  - The number of syllables and the number of words is also a good indicator of the readability of the requirement.
- “The administrator shall be able to generate invoices associated to different companies. He shall be informed of the pending invoices to generate an email and inform them.”

Common errors (7): pronouns

- “The administrator shall be able to generate invoices associated to different companies and he shall be informed of the pending invoices to generate an email to report them.”
  - Too many pronouns make difficult to understand the requirement
  - “He”: Who is “he”? The administrator? The system?
  - “Them”: Who are “them”? Administrators? Clients?
- “The administrator shall be able to generate invoices associated to different companies. The administrator shall be informed of the pending invoices to generate an email to report companies.”

Common errors (8): pseudocode

- “The administrator shall be able to generate invoices associated to different companies. The administrator shall be informed of the pending invoices to generate an email to report companies. The process to find pending invoices is:

  1. Iterate over all invoices
  2. If *Invoice_date* + *Payment_conditions* is greater than *Current_date*, then:
     - If client is of type A, then add 1 month extension
     - Otherwise, while there are pending invoices no new invoices are generated and a new email will be sent each week

  – Avoid the use of pseudocode.

  – Long requirements can imply low-quality requirements.

Common errors (9): number of terms

- “Clients can make order via Internet. These orders shall include a delivery date and a number of products. Once a new order is received, the packaging team shall collect all the products and notify the client via email. HTTP and HTTPS as communication protocols and the Iexplorer and Firefox browsers shall be supported. The minimum resolution for the web user interface shall be 1024x768.”
  - Too main terms in the requirement can indicate:
    - A requirement is addressing and mixing different needs.
    - A requirements contains too much detail.
  - Many verbs in a requirement can imply that different needs are being addressed.

Common errors (10): Subjectivity and negations

- "In my opinion, none of the clients should directly send an order to the packaging team. We proceeded in this manner in a previous Project and the result was not satisfactory."
  - No opinions shall be included in a requirement. Just focus on the functionality.
  - Try to avoid too many negative terms to ease the Reading.
- "The client shall not directly send orders to the packaging team".

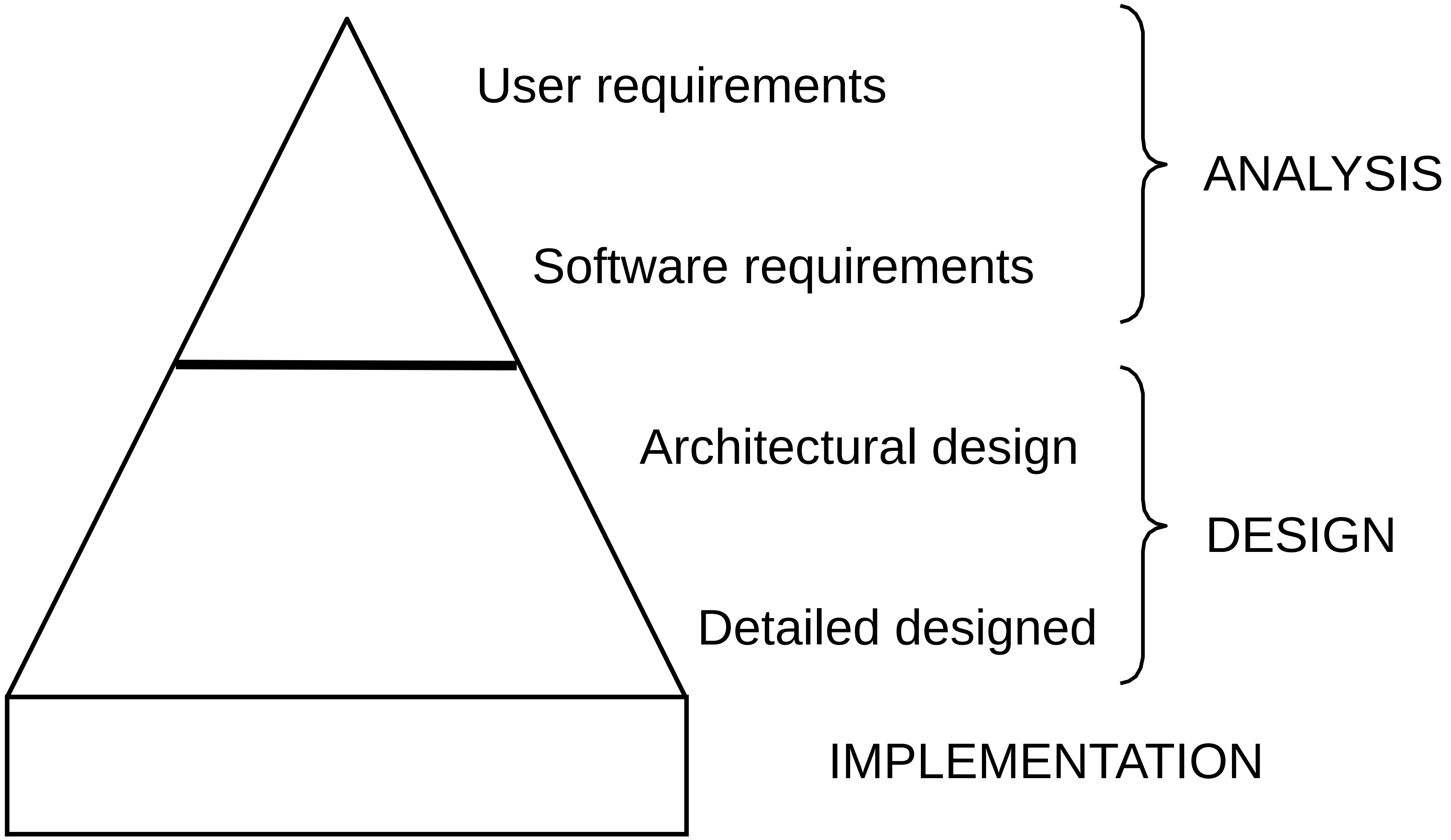
Common Errors (11): lack of precision

- "General speaking, the system shall be able to perform the monitoring process without overloading the server too much".
  - Avoid vague expressions such as "general", "too much"
  - Verify that any restriction can be easily verified
- "The system shall be able to perform the monitoring process in less than 2 seconds and using at the most 250MB of memory"

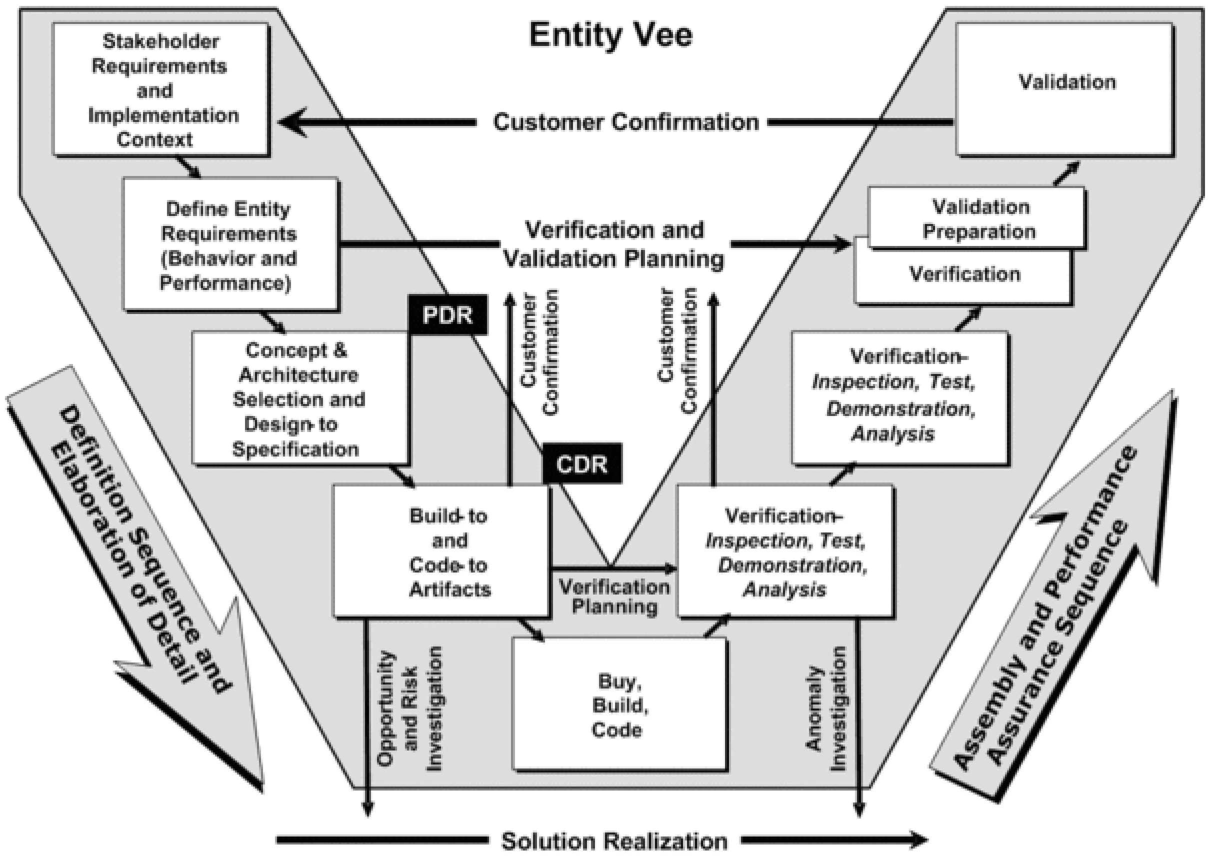
Types of Software Requirements

- Two different levels
  - User requirements and Software requirement
- Classification of requirements
  - **Negative requirements**
  - **Functional requirement**– what? (analysis)
    - Capabilities, information and operational requirements
    - System model: conceptual model + use case model
  - **Non-functional requirements**– restrictions on how? (a kind of pre-design)
    - Special characteristics
    - Restrictions on the capabilities
    - Analysis and documentation
- Traceability: backward / forward

User requirements vs. Software requirements



Example: the Vee model (a software development process)


Entity Vee
Stakeholder Requirements and Implementation Context
Customer Confirmation
Validation
Define Entity Requirements (Behavior and Performance)
Verification and Validation Planning
Validation Preparation
Verification
PDR
Customer Confirmation
Customer Confirmation
Concept & Architecture Selection and Design to Specification
Verification- Inspection, Test, Demonstration, Analysis
CDR
Build-to and Code-to Artifacts
Verification Planning
Verification- Inspection, Test, Demonstration, Analysis
Definition Sequence and Elaboration of Detail
Assembly and Performance Assurance Sequence
Opportunity and Risk Investigation
Buy, Build, Code
Anomaly Investigation
Solution Realization

Source: https://upload.wikimedia.org/wikipedia/en/thumb/5/5d/Entity_Vee_Model.gif/600px-Entity_Vee_Model.gif

User Requirements-Software Requirements: main differences

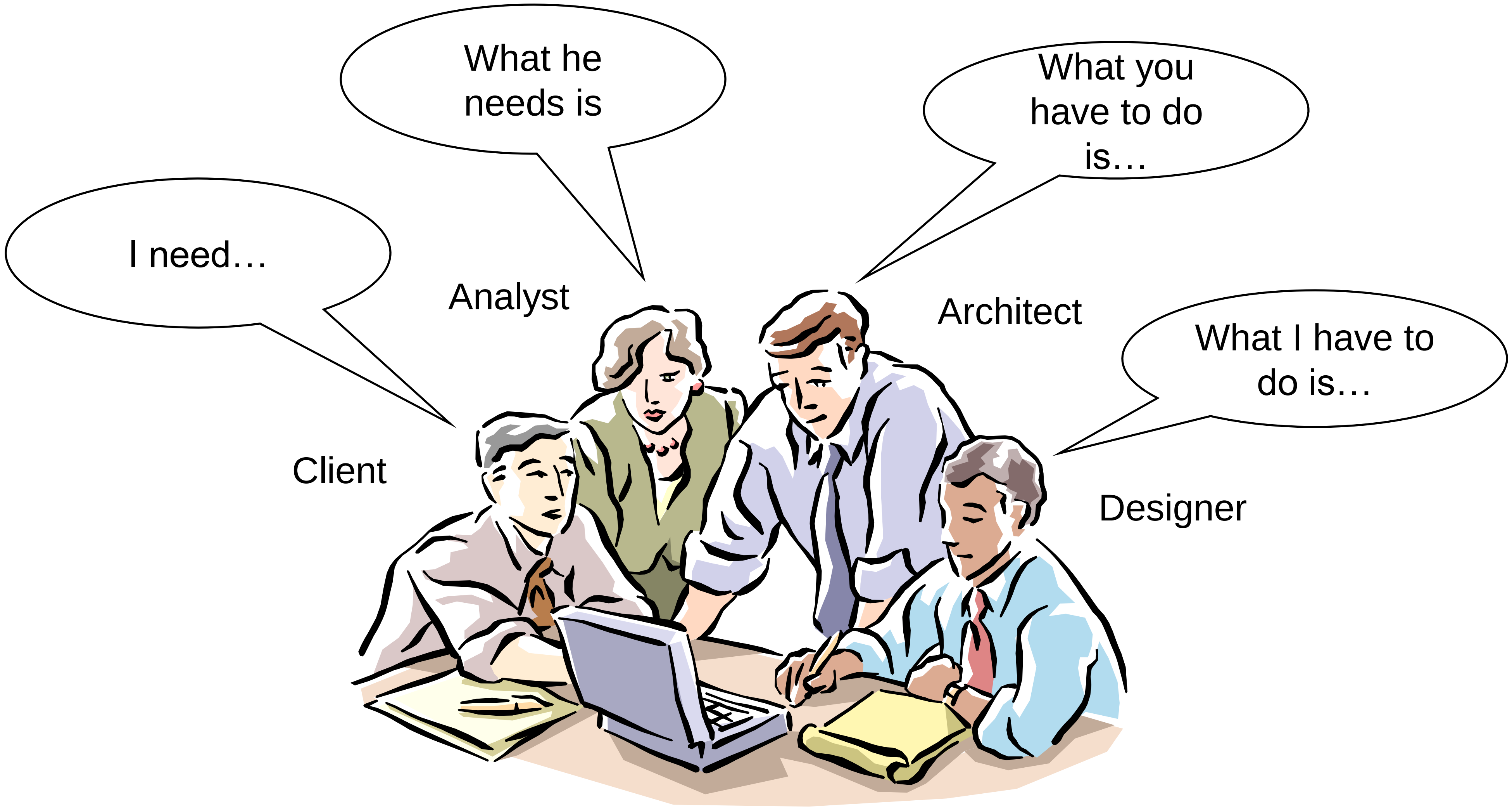
	User Requirements	Software Requirements
Goal	Problem statement Requirements elicitation	Problem description and refinement Requirements analysis
Source	user/client	User/client and analyst
Responsible	user/client	analyst
Audience	user/client (and developer)	developer (and user/client)
Focus on	Product perspective User needs Operational environment Use of prototypes	Domain knowledge Models and formal methods Organization and structure to get consistency and completeness

Workflow vs Documents

Workflow		Document	
USDP	**Traditional**	**IEEE**	**ESA**
Requirements	Requirements Analysis	Software Requirements Specification (IEEE 830-1993)	User Requirements Document
Analysis			Software Requirements Document
Design	Architectural design	Software Design Document (IEEE 1016-1987)	Architectural Design Document
	Detailed design		Detailed Design Document

+ Implementation, Testing, Maintenance...

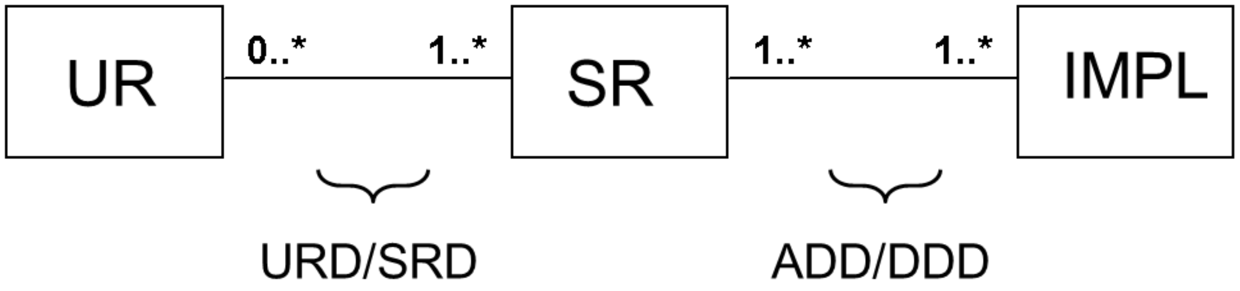
Different points of view– Different documents



Non-functional requirements

- Resource consumption
  - Memory, network, etc.
- Performance
  - Response time, throughput...
- Reliability and availability
  - Mean time between failures, etc.
- Error management
  - Exceptions, failures, bugs
- Interface requirement
  - User experience, interoperability
- Restrictions
  - Standards, technical restrictions, etc.
- Safety and Security
  - System security (*logical*), Physical or personal safety (*person*)

Traceability matrix



Sparse matrix

	UR1	UR2	UR3
SR1	X		
SR2	X		X
SR3		X	
SR4			X
SR5			X

Three-column matrix

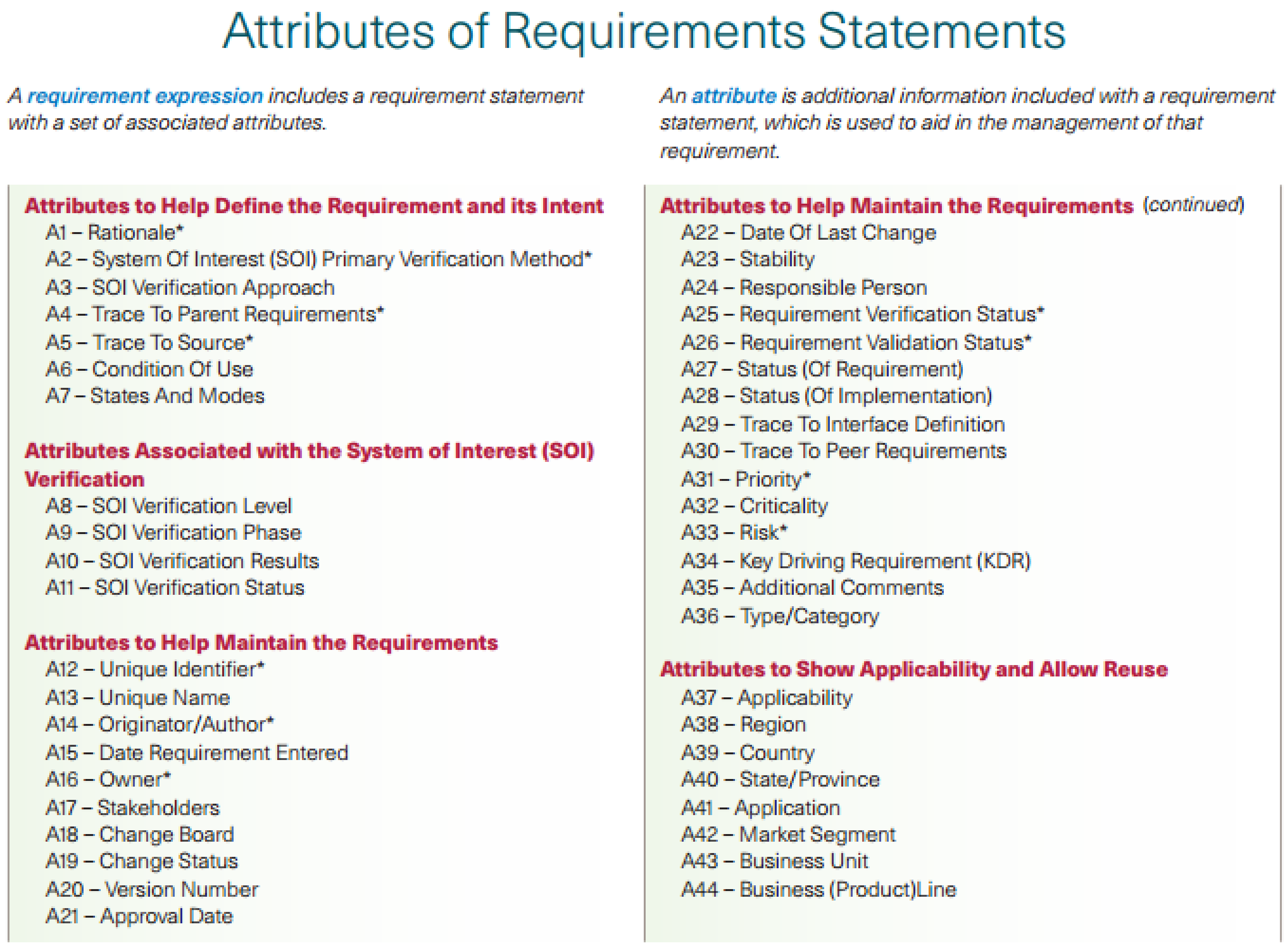
UR	SR	Description
1	1,2	
2	3	
3	2,4,5	

SR	UR	Title
1	1	
2	1,3	
3	2	
4	3	
5	3	

Properties and attributes of Software Requirements

- Global desired properties
  - Completeness
    - Organization by type of requirement
  - Consistency
    - Cross-reference matrix
- Individual desired properties
  - Size
  - Clarity
  - Verifiability: validation and verification
  - Error conditions
- Attributes
  - Automatic: identifier, creator, creation date…
  - Mandatory: type, status, description (short).
  - Optional: description, source, need, priority, stability, complexity, risk, cost…

INCOSE-Guide for Writing Requirements, 2015



Attributes of Requirements Statements

*A **requirement expression** includes a requirement statement with a set of associated attributes.*

*An **attribute** is additional information included with a requirement statement, which is used to aid in the management of that requirement.*

Attributes to Help Define the Requirement and its Intent

- A1 – Rationale*
- A2 – System Of Interest (SOI) Primary Verification Method*
- A3 – SOI Verification Approach
- A4 – Trace To Parent Requirements*
- A5 – Trace To Source*
- A6 – Condition Of Use
- A7 – States And Modes

Attributes Associated with the System of Interest (SOI) Verification

- A8 – SOI Verification Level
- A9 – SOI Verification Phase
- A10 – SOI Verification Results
- A11 – SOI Verification Status

Attributes to Help Maintain the Requirements

- A12 – Unique Identifier*
- A13 – Unique Name
- A14 – Originator/Author*
- A15 – Date Requirement Entered
- A16 – Owner*
- A17 – Stakeholders
- A18 – Change Board
- A19 – Change Status
- A20 – Version Number
- A21 – Approval Date

Attributes to Help Maintain the Requirements *(continued)*

- A22 – Date Of Last Change
- A23 – Stability
- A24 – Responsible Person
- A25 – Requirement Verification Status*
- A26 – Requirement Validation Status*
- A27 – Status (Of Requirement)
- A28 – Status (Of Implementation)
- A29 – Trace To Interface Definition
- A30 – Trace To Peer Requirements
- A31 – Priority*
- A32 – Criticality
- A33 – Risk*
- A34 – Key Driving Requirement (KDR)
- A35 – Additional Comments
- A36 – Type/Category

Attributes to Show Applicability and Allow Reuse

- A37 – Applicability
- A38 – Region
- A39 – Country
- A40 – State/Province
- A41 – Application
- A42 – Market Segment
- A43 – Business Unit
- A44 – Business (Product)Line

Consistency: conflicts, overlapping sand redundancies

	R1	R2	R3	R4	R5	R6	R7
R1			+		x	x	
R2							
R3	+			+		+	
R4			+		x		o
R5	x			x			
R6	x		+				
R7				o			

Conflict (x)	**Overlapping (+)**	**Redundancy (o)**	**Independent**
R1 and R5 R1 and R6 R4 and R5	R1 and R3 R3 and R4 R3 and R6	R4 andR7 (→R7xR5, R7+R3)	R2

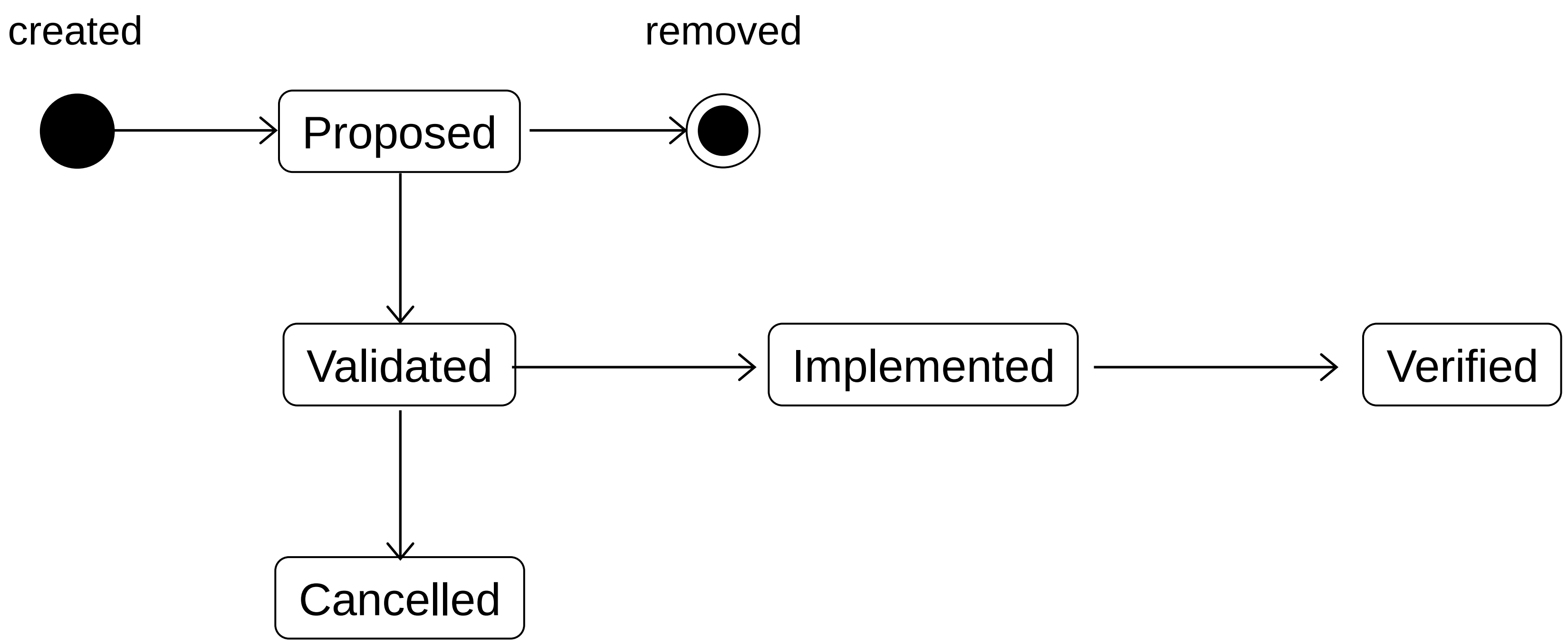
Requirement template (example)

ID	007
Title	Guest users
Author	Ana García (analyst)
Creation Date	05-09-2011
Source	Juan Gómez (ICT director)
Type	Functional
Necessity	Essential
Status	Proposed
Description	The guest users are not registered users: they can Access to all functionalities for non-registered users without making any log in and they have an unlimited number of opened sessions.
Test plan	1. Check that all functionalities for guest users are opened. 2. Check that the number of opened sessions for guest users is unlimited.

Methods to organize requirements

- The aforementioned techniques…
  - Traceability matrix
  - Consistency matrix: conflicts, overlappings and redundancies
  - Modularity and nested requirements: topics or thematic areas
- Organize requirements according to the system model
  - Use case model
    - Referred classes
    - Referred operations
  - Conceptual model (classes)
    - Attributes
    - Operations
  - It is COMPLETELY NECESSARY to ensure consistency between models and requirements
- Requirements lifecycle
- Use of requirements management system (RMS) to analyze and organize

Requirement Lifecycle



Features of a Requirements Management System (1)

- Support to multiple projects and types of users
  - User profiles: analysts, Project leaders, administrators,.
  - Permission management: read and write
- Project configuration and management
  - Global properties: name, description, location, etc.
  - Specific requirements attributes (per type, etc.)
- Access, session and versioning
  - Automatic session management (baselines and streams): starting and finishing time, changes, etc.
  - Versioning: NO, PARTIAL, COMPLETE.
  - Change notification (optional).
  - Diff.
- Requirements properties
  - Group in packages and sub packages (modules)
  - Attributes: automatic, mandatory, optional.
  - Requirements links: navigability and asymmetry. “Suspected” requirements.
  - Other artefacts: scenarios, models, etc.

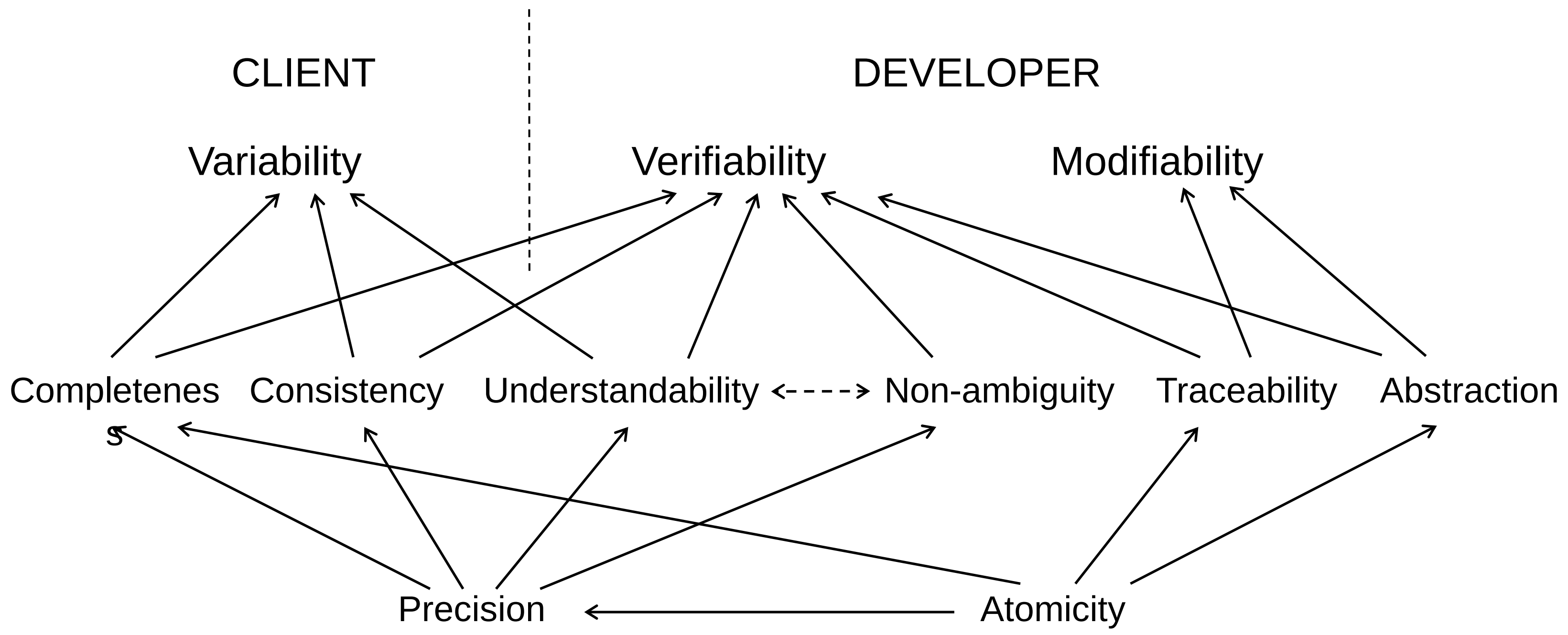
Features of a Requirements Management System (2)

- Visualization, browsability and edition
  - Configurable card or table to show attributes.
  - Sorting and filtering capabilities.
  - Search and replace.
  - Module navigability.
  - Duplicated requirements.
- Reporting capabilities
  - Requirements list: sorting, filtering, visible attributes...
  - Stats: evolution, conflicts, etc.
  - Traceability and consistency matrix.
- Utilities
  - Writing assistance to create glossaries and requirements.
  - Ensuring coherence between requirements and models.
  - Automatic detection of overlapped requirements.
  - Quality metrics.

Quality Metrics in Software Requirements

- Why?
  - High-quality from the very early stage of the project
  - People involved in the process (quality managers)
  - Added cost
- Quality metrics: How good are requirements?
  - What we have to measure…
    - Desired properties (qualitative assessment)
    - Measurable indicators (quantitative assessment)
  - How we can measure…
    - Hand-made metrics (questionnaire-based review)
    - Automatic metrics (linguistic features)
- Quality metrics: How effective is the process?
  - Measures to stablish the effectiveness of inspecting requirements
  - Measures to stablish the effectiveness of the requirements analysis process

Properties and quality metrics



Morphology	Size, Legibility, Punctuation, Acronyms and Abbreviations
Lexical	Negative terms, connectors, ambiguity, etc.
Syntax	Spellchecking, grammar, passive or active voice, conditional or imperative clauses, domain terms
Relational	Number of versions, nesting degree, dependencies, overlapping, etc.