Requirements Specification

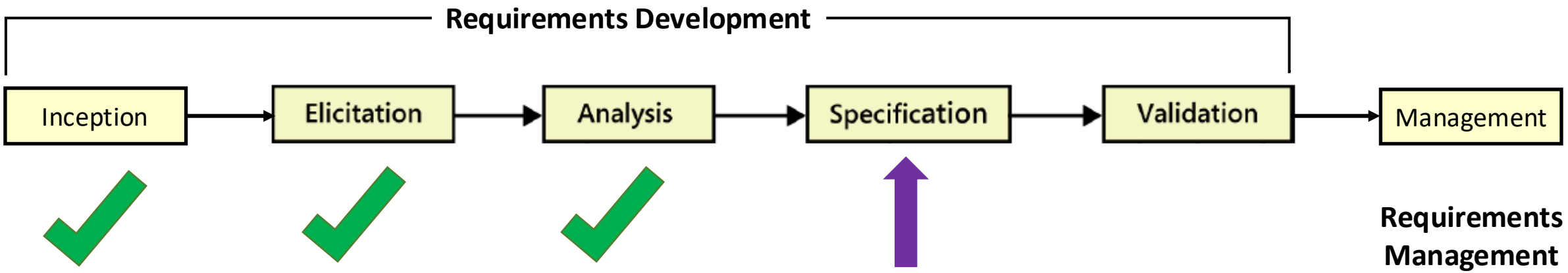
"*Hi, Gautam. This is Ruth calling from the Austin branch. We got that latest drop of the website software for the online music store. I wanted to ask you about the song preview feature. That's not working the way I had in mind*."

Gautam replied, "*Let me find the requirements you sent for that. Here they are. The user story said, 'As a Customer, I want to listen to previews of the available songs so I can decide which ones to buy.' My notes say that when we discussed this, you said each song sample should be 30 seconds long and that it should use our built-in MP3 player so the customer didn't have to wait for another player to launch. Isn't that working correctly?*"

"*Well, yes, that all works fine,*" said Ruth, "*but there are a couple of problems. I can click on the play icon to start the sample, but I don't have any way to pause it or stop it. I'm forced to listen to the entire 30-second sample. Also, all the samples start at the beginning of the song. Some songs have long introductions so you really can't hear what they're like from just the beginning. The sample should start somewhere in the middle of those songs so people could hear what they're really like. And the sample starts playing at full volume and then stops abruptly. If the customer's speakers are up pretty loud this could be startling. I think it would be better if we fade in and fade out on each sample.*"

Gautam was a little frustrated. "*I wish you had told me all of this when we spoke earlier. You didn't give me much to go on so I just had to make my best guess. I can do all that, but it's going to take a few more days.*"

RE Process


Requirements Development
Inception
Elicitation
Analysis
Specification
Validation
Management
Requirements Management

Characteristics of Excellent Requirements
Complete 🏆

Each requirement must contain all the information necessary for the reader to understand it.

- Functional requirements need to have all the information necessary for the developer to implement it correctly.
- Use TBD (to be determined) as a standard flag to highlight any gaps
- Resolve all TBDs before developers proceed with implementation

Characteristics of Excellent Requirements
Correct 🏆

- Each requirement must accurately describe a capability that will meet some stakeholder's need and must clearly describe the functionality to be built.
- You'll have to go to the source of the requirement to check its correctness – a user who supplied the initial user requirement, a higher-level system requirement, a use case, a business rule, or another document.
  - E.g., a low-level requirement that conflicts with its parent is not correct.

Characteristics of Excellent Requirements
Feasible 🏆

- It must be possible to implement each requirement given:
  - Capabilities and limitations of the system and its operating environment
  - Project constraints of time, budget, and staff
- During elicitation, developers can provide a reality check on what can and cannot be done technically and what can be done only at excessive cost or effort
- Incremental development approaches and proof-of-concept prototypes are two ways to evaluate requirement feasibility.

Characteristics of Excellent Requirements
Necessary 🏆

- Each requirement should describe a capability that:
  - Provides stakeholders with anticipated business value
  - Differentiates the product in the marketplace
  - Is required for conformance to an external standard, policy, or regulation
- If someone asks why a particular requirement is included, there should be a good answer

Characteristics of Excellent Requirements
Prioritized 🏆

- Prioritize business requirements according to which are most important to achieving the desired value.
- If all requirements are equally important, the project manager doesn't know how best to respond to schedule overruns, personnel losses, or new requirements that come along.
- Requirements prioritization should be a collaborative activity involving multiple stakeholder perspectives.

Characteristics of Excellent Requirements
Unambiguous 🏆

- Natural language is prone to two types of ambiguity:
  - One can think of more than one interpretation for a given requirement
  - Different people have different interpretations for requirements
- "*Comprehensible*" is related to "*unambiguous*" – readers must understand what each requirement is saying
- You'll never remove all the ambiguity from requirements—that's the nature of human language

Mitigation

Use a formal peer review among colleagues to compare each others' understanding of requirements. This will clean up a lot of the wors issues.

Characteristics of Excellent Requirements
Verifiable 🏆

- A requirement is verifiable if a tester can devise tests to determine whether or not it was properly implemented
- If a requirement isn't verifiable, deciding whether it was correctly implemented becomes a matter of opinion, not objective analysis
- Requirements that are incomplete, inconsistent, infeasible, or ambiguous are also unverifiable
- Testers are good at examining requirements for verifiability. Include them in your requirements peer reviews to catch problems early.

Characteristics of Requirements Collections
🏆 *Complete* 🏆

- There are always some assumed or implied requirements, although they carry more risk than explicitly stated requirements
- Any specification that contains TBDs is incomplete

Characteristics of Requirements Collections
🏆 *Consistent* 🏆

- Consistent requirements don’t conflict with other requirements of the same type or with higher-level business, user, or system requirements
- If BA doesn’t resolve conflicts among requirements, developers will have to deal with them
- Tip: record the originator of each requirement to go back to in case of conflict

Characteristics of Requirements Collections
🏆 *Modifiable* 🏆

To make your SRS modifiable:

- maintain a history of changes made to each requirement, especially after they are baselined
- Learn about connections and dependencies between requirements so you can find all the ones that must be changed together
- Use unique labels for each requirement to make them easily identifiable
- Avoid redundancy – repeating a requirement in multiple places where it logically belongs makes the document easier to read but harder to maintain
- Cross-reference related items in the SRS to help keep them synchronized when making changes

Characteristics of Requirements Collections

🏆 *Traceable* 🏆

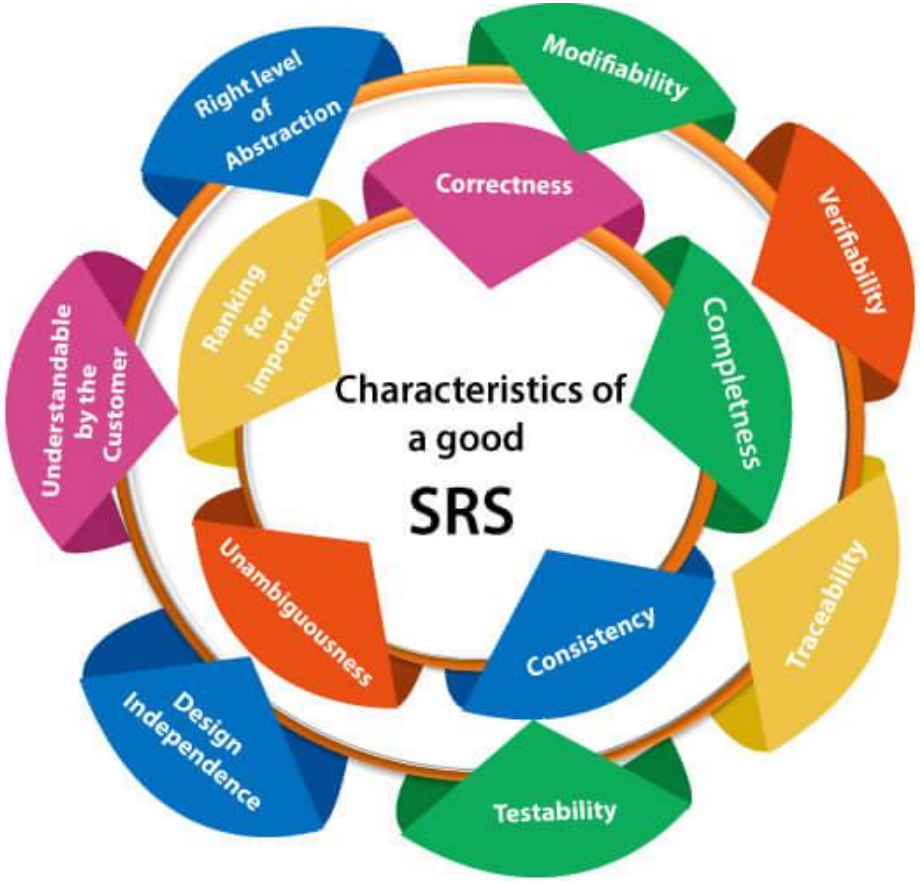
A traceable requirement can be linked both backward to its origin and forward to derived requirements, design elements, code that implements it, and tests that verify its implementation

Unique identifiers are important here as well

Avoid combining multiple requirements together into a single statement, because the different requirements might trace to different development components

Characteristics of Reuirements Collections

- You’re never going to create a perfect specification in which all requirements demonstrate all of these ideal attributes. But if you keep these characteristics in mind when you write and review the requirements, you’ll produce better requirements specifications and better software



Guideline for Writing Requirements

- There isn’t a formula for writing excellent requirements – the best teachers are experience and feedback from the recipients of your requirements
- Receiving constructive feedback from colleagues with sharp eyes is a great help, hence peer reviews
- Two important goals of writing requirements:

  - Anyone who reads the requirement comes to the same interpretation as any other reader
  - Each reader’s interpretation matches what the author intended to communicate

Guidelines for Requirements

1- System or user perspective

2- Writing style

3- Level of detail

4- Representation techniques

5- Avoiding ambiguity

6- Avoiding incompleteness

Find me under the discussion of *Requirements Analysis*

1- System or User Perspective

- You can write functional requirements from the perspective of either something the system does or something the user can do
- Effective communication is the overarching goal, so it's fine to intermingle these styles, using whichever is clearer
    1. State requirements in a consistent fashion, such as "*The system shall*" or "*The user shall*," followed by an action verb, followed by the observable result
    2. Specify the trigger action or condition that causes the system to perform the specified behavior

1- System or User Perspective

Easy Approach to Requirements Syntax

[optional precondition] [optional trigger event] the system shall

[expected system response]

Precondition

No trigger

E.g., *If the requested chemical is found in the chemical stockroom, the system shall display a list of all containers of the chemical that are currently in the stockroom.*

Because this requirement is describing system behavior, some requirement writers argue that there's no need to repetitively say "the system shall". However, including the "shall" and writing in the active voice makes it clear what entity is taking the action described.

1- System or User Perspective

The system shall let (allow, permit, or enable) the [a particular user class or name] to [do something]

The [user class or actor name] shall be able to [do something] [to some object] [qualifying conditions, response time, or quality statement]

User class in place "user" – reduces misinterpretation

E.g., The Chemist shall be able to reorder any chemical he has ordered in the past by retrieving and editing the order details

2- Writing Style

Helps those who are skimming through a document

- Adjust your writing style to put the punch line—statement of need or functionality—first, followed by supporting details (rationale, origin, priority, and other requirement attributes)
- Use tables, structures lists, diagrams, and other visual elements to help break the unvarying tone of functional requirements and provide richer communication to those who learn best in different ways

2- Writing Style

- SRS is not the place to practice creative writing skills
  - Avoid interleaving passive and active voice in an attempt to make the material more interesting to read.
  - Don't use multiple terms for the same concept just to achieve variety (customer, account, patron, user, client).
- Being easy to read and understand is an essential element of well-written requirements; being interesting is, frankly, less important

2- Writing Style - Tips

Clarity and conciseness

- Use complete sentences, proper grammer, spelling and punctuation
- Keep sentences and paragraphs short and direct – avoid jargon
- Define specialized items in glossary
- Write concisely – "*needs to provide the user with the capability to*" can be condensed to "*shall*"
- But remember, clarity is more important than conciseness

2- Writing Style - Tips

The keyword "*shall*"

- Objectors say "*that's not how people talk*"
- You might prefer to say "*must*," "*needs to*," or something similar, but be consistent

2- Writing Style - Tips

Active voice

- Write in the active voice to make it clear what entity is taking the action described

- ”*Upon product upgrade shipment, the serial number will be updated on the contract line*”

Recipient **Passive**

Performer?

Will the system do that automatically, or is the user expected to update the serial number?

- ”*When Fulfillment confirms that they shipped a product upgrade, the system shall update the customer’s contract with the new product serial number*”

2- Writing Style - Tips

Individual requirements

- Avoid writing long narrative paragraphs that contain multiple requirements
- Clearly distinguish individual requirements from background or contextual information
- Words such as “*and*,” “*or*,” “*additionally*,” or “*also*” in a requirement suggest that several requirements might have been combined – you don’t necessarily have to abandon “*and”*, but make sure the conjunction is joining two parts of a single requirement

2- Writing Style - Tips

Individual requirements

- Avoid using "*and/or*" in a requirement; it leaves the interpretation up to the reader
  - "*The system must permit search by order number, invoice number, and/or customer purchase order number*" – This requirement would permit the user to enter one, two, or three numbers at once when performing a single search. That might not be what's intended.

2- Writing Style - Tips

Individual requirements

- The words “*unless*,” “*except*,” and “*but*” also indicate the presence of multiple requirements

  - “*The Buyer’s credit card on file shall be charged for payment, unless the credit card has expired*”
  - Failing to specify what happens when the “unless” clause is true is a common source of missing requirements.

2- Writing Style - Tips

Individual requirements

- The words “*unless*,” “*except*,” and “*but*” also indicate the presence of multiple requirements
  - Split into two requirements to address the behavior for the two conditions of the credit card being active and expired

  - “*If the Buyer’s credit card on file is active, the system shall charge the payment to that card.*” and “*If the Buyer’s credit card on file has expired, the system shall allow the Buyer to either update the current credit card information or enter a new credit card for payment.*”

3- Level of Detail

Appropriate detail

- There's no single correct answer to the commonly asked question, "*How detailed should the requirements be?*"
- Provide enough specifics to minimize the risk of misunderstanding, based on the development team's knowledge and experience
- The fewer the opportunities for discussion about requirements issues, the more specific you need to be
- If a developer can think of several possible ways to satisfy a requirement and all are acceptable, the specificity and detail are about right

3- Level of Detail

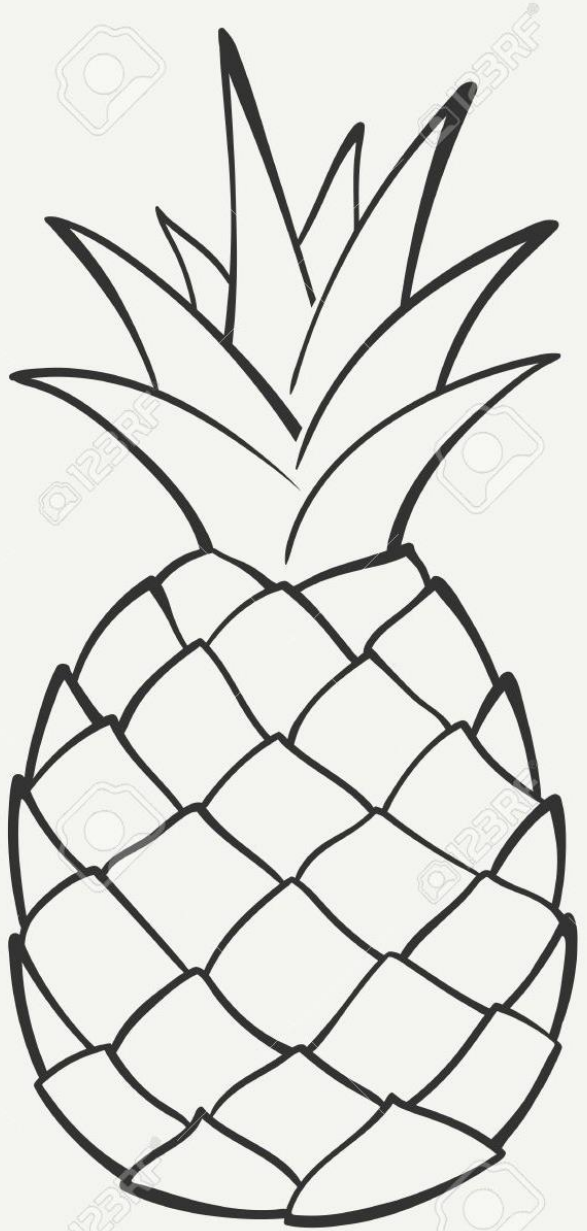
Appropriate detail – you should include *more* detail when:

- The work is being done for an external client
- Development or testing will be outsourced
- Project team members are geographically dispersed
- System testing will be based on requirements
- Accurate estimates are needed
- Requirements traceability is needed

3- Level of Detail

Appropriate detail – you should include *less* detail when:

- The work is being done internally for your company.
- Customers are extensively involved.
- Developers have considerable domain experience.
- Precedents are available, as when a previous application is being replaced.
- A package solution will be used.



3- Level of Detail

Consistent granularity

- It’s not necessary to specify all of your requirements to the same level of detail – you might go into more depth in an area that presents higher risk than others
- Within a set of related requirements, though, it’s a good idea to try to write functional requirements at a consistent level of granularity

3- Level of Detail

Consistent granularity

- Helpful guideline: write individually testable requirements
- If you can think of a small number of test cases to verify that a requirement was correctly implemented, it's probably at an appropriate granularity
- If you envision numerous and diverse tests, perhaps several requirements are combined and ought to be separated

3- Level of Detail

Consistent granularity

- Very fine grained
- Will need few tests for verification of correct behavior
- Better expressed in the form of a table

- *"The system shall interpret the keystroke combination Ctrl+S as File Save."* and

  "The system shall interpret the keystroke combination Ctrl+P as File Print."

- *"The product shall respond to editing directives entered by voice."*

An entire product on it's own!

4- Representation Techniques

- Some alternatives to the natural language requirements are lists, tables, visual analysis models, charts, mathematical formulas, photographs, sound clips, and video clips
- These won’t suffice as substitutes for written requirements in many cases, but they serve as excellent supplemental information to enhance the reader’s understanding.

4- Representation Techniques

- Example:

  The Text Editor shall be able to parse <format> documents that define <jurisdiction> laws.

  - 3 possible values for *<format>* and 4 possible values for *<jurisdiction>*, for a total of *12* similar requirements
  - The SRS did indeed contain 12 such requirements, but one of the combinations was missing and another was duplicated.
  - Prevent such errors by representing them in a table, which is more compact and less boring than a requirements list.

4- Representation Techniques

- The generic requirement could be stated as:

  Editor.DocFormat *The Text Editor shall be able to parse documents in several formats that define laws in the jurisdictions shown in Table 11-1.*

Cell contents indicate suffix to append to master requirement's identifier

Jurisdiction	*Tagged format*	*Untagged format*	*ASCII format*
Federal	*.1*	*.2*	*.3*
State	*.4*	*.5*	*.6*
Territorial	*.7*	*N/A*	*.8*
International	*.9*	*.10*	*.11*

N/A indicates combination doesn't have a functional requirement for some logical reason.

- Clearer than omitting the irrelevant combination from a long list and having a reader wonder where it is
- Ensures completeness in requirements sets

Editor.DocFormat.3 *The Text Editor shall be able to parse ASCII documents that define federal laws*

5- Avoiding Ambiguity

- Requirements quality is in the eye of the reader, not the author
- If a reader has questions, the requirement needs additional work
- Peer reviews are the best way to find places where the requirements aren't clearly understood by all the intended audiences
- Sources of ambiguity:
  - Fuzzy words
  - The A/B construct
  - Boundary values
  - Negative requirements

5- Avoiding Ambiguity – *Sources*

Ambiguous language leads to unverifiable requirements

Fuzzy words

- Use terms consistently and as defined in the glossary
- Watch out for synonyms and near-synonyms
- Avoid vague and subjective terms
- If you use a pronoun to refer to something mentioned earlier, make sure the antecedent is crystal clear
- Adverbs introduce subjectivity and hence ambiguity. Avoid words like *reasonably, appropriately, generally, approximately, usually, systematically, and quickly* because the reader won't be sure how to interpret them.

5- Avoiding Ambiguity – *Sources*

Table 11-2. Some ambiguous terms to avoid in requirements

Ambiguous terms	Ways to improve them
acceptable, adequate	Define what constitutes acceptability and how the system can judge this.
and/or	Specify whether you mean "and," "or," or "any combination of" so the reader doesn't have to guess.
as much as practicable	Don't leave it up to the developers to determine what's practicable. Make it a TBD and set a date to find out.
at least, at a minimum, not more than, not to exceed	Specify the minimum and maximum acceptable values.
best, greatest, most	State what level of achievement is desired and the minimum acceptable level of achievement.
between, from X to Y	Define whether the end points are included in the range.
depends on	Describe the nature of the dependency. Does another system provide input to this system, must other software be installed before your software can run, or does your system rely on another to perform some calculations or provide other services?
efficient	Define how efficiently the system uses resources, how quickly it performs specific operations, or how quickly users can perform certain tasks with the system.
fast, quick, rapid	Specify the minimum acceptable time in which the system performs some action.
flexible, versatile	Describe the ways in which the system must be able to adapt to changing operating conditions, platforms, or business needs.
i.e., e.g.	Many people are unclear about which of these means "that is" (i.e., meaning that the full list of items follows) and which means "for example" (e.g., meaning that just some examples follow). Use words in your native language, not confusing Latin abbreviations.
improved, better, faster, superior, higher quality	Quantify how much better or faster constitutes adequate improvement in a specific functional area or quality aspect.
including, including but not limited to, and so on, etc., such as, for instance	List all possible values or functions, not just examples, or refer the reader to the location of the full list. Otherwise, different readers might have different interpretations of what the

5- Avoiding Ambiguity – *Sources*

	whole set of items being referred to contains or where the list stops.
in most cases, generally, usually, almost always	Clarify when the stated conditions or scenarios do not apply and what happens then. Describe how either the user or the system can distinguish one case from the other.
match, equals, agree, the same	Define whether a text comparison is case sensitive and whether it means the phrase "contains," "starts with," or is "exact." For real numbers, specify the degree of precision in the comparison.
maximize, minimize, optimize	State the maximum and minimum acceptable values of some parameter.
normally, ideally	Identify abnormal or non-ideal conditions and describe how the system should behave in those situations.
optionally	Clarify whether this means a developer choice, a system choice, or a user choice.
probably, ought to, should	Will it or won't it?
reasonable, when necessary, where appropriate, if possible, as applicable	Explain how either the developer or the user can make this judgment.
robust	Define how the system is to handle exceptions and respond to unexpected operating conditions.
seamless, transparent, graceful	What does "seamless" or "graceful" mean to the user? Translate the user's expectations into

	specific observable product characteristics.
several, some, many, few, multiple, numerous	State how many, or provide the minimum and maximum bounds of a range.
shouldn't, won't	Try to state requirements as positives, describing what the system will do.
state-of-the-art	Define what this phrase means to the stakeholder.
sufficient	Specify how much of something constitutes sufficiency.
support, enable	Define exactly what functions the system will perform that constitute "supporting" some capability.
user-friendly, simple, easy	Describe system characteristics that will satisfy the customer's usage needs and usability expectations.

5- Avoiding Ambiguity – *Sources*

The A/B construct

- Expression in the form "A/B," in which two related (or synonymous, or opposite) terms are combined with a slash
  - *"The system shall provide automated information collection of license key data for a mass release from the Delivery/Fulfillment Team*."

Possible interpretations

The name of the team is Delivery/Fulfillment.
Delivery and fulfillment are synonyms.
Either the Delivery Team or the Fulfillment Team can do a mass release, so the slash means "or."
Some projects call the group a Delivery Team; others call it a Fulfillment Team.
The Delivery Team and the Fulfillment Team jointly do a mass release, so the slash means "and."

5- Avoiding Ambiguity – *Sources*

Boundary values

"Vacation requests of up to 5 days do not require approval. Vacation requests of 5 to 10 days require supervisor approval. Vacation requests of 10 days or longer require management approval."

What happens to vacations of exactly 5 and 10 days?

It gets even more confusing if fractions are involved!

5- Avoiding Ambiguity – *Sources*

Boundary values

- Use “*through*,” “*inclusive*,” and “*exclusive*” to denote whether the endpoints of the numerical range lie inside or outside the range

“Vacation requests of 5 or fewer days do not require approval. Vacation requests of longer than 5 days through 10 days require supervisor approval. Vacation requests of longer than 10 days require management approval.”

5- Avoiding Ambiguity – *Sources*

Negative requirements – requirements that say what the system will not do rather than what it will do

- How do you implement a don't-do-this requirement?
- Double and triple negatives are particularly tricky to decipher!

- "*Prevent the user from activating the contract if the contract is not in balance.*"

- "*The system shall allow the user to activate the contract only if the contract is in balance.*"

5- Avoiding Ambiguity – *Sources*

Negative requirements

- If you need to indicate that a certain functionality is out of scope, include the restriction in the Limitations and Exclusions section of the vision and scope document instead of listing it as a negative requirement

Sample Requirements, before and after

Example 1 *The Background Task Manager shall provide status messages at regular intervals not less than every 60 seconds.*

Sample Requirements, before and after

Example 1 *The Background Task Manager shall provide status messages at regular intervals not less than every 60 seconds.*

- What are the status messages?
- Under what conditions and in what fashion are they provided to the user?
- If displayed on the screen, how long do they remain visible?
- Is it okay if they just flash up for half a second?
- The timing interval is not clear, and the word “every” just muddles the issue

Sample Requirements, before and after

Example 1 *The Background Task Manager shall provide status messages at regular intervals not less than every 60 seconds.*

- Is the interval between status messages supposed to be at least 60 seconds, so providing a new message once per year is okay?
- If the intent is to have at most 60 seconds elapse between messages, would one millisecond be too short an interval?

Because of these problems, this requirement is not verifiable

Sample Requirements, before and after

One way to rewrite it (after we get some information from the customer would be)**:**

1. The Background Task Manager (BTM) shall display status messages in a designated area of the user interface.

1.1. The BTM shall update the messages every 60 plus or minus 5 seconds after background task processing begins.

1.2. The messages shall remain visible continuously during background processing.

1.3. The BTM shall display the percent of the background task that is completed.

1.4. The BTM shall display a "Done" message when the background task is completed.

1.5. The BTM shall display a message if the background task has stalled.

Sample Requirements, before and after

- Rewriting a flawed requirement often makes it longer because information was missing
- Splitting this into multiple child requirements makes sense because each will demand separate tests, and also each one will be individually traceable
- If there are other status messages that the BTM might display, list them in a table of conditions and corresponding messages. If they’re documented elsewhere in the document, reference them instead of replicating them.

Sample Requirements, before and after

1. The Background Task Manager (BTM) shall display status messages in a designated area of the user interface.

Defers the placement of the messages to being a design issue

If you specify the display location in the requirements, it becomes a **design constraint** placed on the developer. Unnecessarily constrained design options frustrate the programmers and can result in a suboptimal product design.

If we were adding this to an existing appliction that has a status bar, where users are accustomed to seeing status messages, it would make perfect sense to specify that in the SRS.

Sample Requirements, before and after

Example 2 *Corporate project charge numbers should be validated online against the master corporate charge number list, if possible.*

Sample Requirements, before and after

Example 2 *Corporate project charge numbers should be validated online against the master corporate charge number list, if possible.*

- The phrase “if possible” is ambiguous
  - “*if it’s technically feasible”*? (a question for the developer)
  - “if the master charge number list can be accessed at run time”?
- If you aren’t sure whether a requested capability can be delivered, use TBD to indicate that the issue is unresolved. After investigation, either the TBD goes away or the requirement goes away.

Sample Requirements, before and after

Example 2 *Corporate project charge numbers should be validated online against the master corporate charge number list, if possible.*

- Doesn't specify what to do when the validation either passes or fails
- The word "*should*" is imprecise

Sample Requirements, before and after

One way to rewrite it:

"At the time the requester enters a charge number, the system shall display an error message if the charge number is not in the master corporate charge number list."

- A related requirement would address the exception condition of the master corporate charge number list not being available at the time the validation was attempted.

Sample Requirements, before and after

Example 3 *The device tester shall allow the user to easily connect additional components, including a pulse generator, a voltmeter, a capacitance meter, and custom probe cards.*

(This requirement is for a product containing embedded software that's used to test several kinds of measurement devices)

Sample Requirements, before and after

Example 3 *The device tester shall allow the user to easily connect additional components, including a pulse generator, a voltmeter, a capacitance meter, and custom probe cards.*

- The word “easily” implies a usability requirement, but it is neither measurable nor verifiable
- “Including” doesn’t make it clear whether this is the complete list of external devices that must be connected to the tester or if there’s more

Sample Requirements, before and after

One way to rewrite it:

1. *The device tester shall incorporate a USB port to allow the user to connect any measurement device that has a USB connection*.
2. *The USB port shall be installed on the front panel to permit a trained operator to connect a measurement device in 10 seconds or less*.

Note: The preceding requirements contain some intentional design constraints. The BA shouldn't make such decision on their own, however. Instead, detect the flawed requirements and discuss them with the appropriate stakeholders so they can be clarified.

Sample Requirements, before and after

TRAP

Watch out for analysis paralysis. All of the sample “after” requirements can be improved further, but you can’t spend forever trying to perfect the requirements. Remember, your goal is to write requirements that are good enough to let your team proceed with design and construction at an acceptable level of risk.