1. **PBR Procedure from the Tester’s Perspective**

For each requirement, generate a test or set of test cases that let you ensure that a system implementation satisfies the requirement. Follow the procedure below to generate the test cases, using the questions provided to identify faults in the requirements.

General questions: Read each requirement once and record the number and page along with the inputs to the requirement.

* Q1. Does the requirement make sense from what you know about the application or from what is specified in the general description?
* Q2. Do you have all the information necessary to identify the inputs to the requirement? Based on the general requirements and your domain knowledge, are these inputs correct for this requirement?
* Q3. Have any of the necessary inputs been omitted?
* Q4. Are any inputs specified that are not needed for this requirement?
* Q5. Is this requirement in the appropriate section of the document?

***Part a: Building equivalence sets.***

For each input, divide the input domain into sets of data (called equivalence sets); all values in each set will cause the system to behave similarly. Determine the equivalence sets for a particular input by understanding the sets of conditions that affect the requirement’s behavior. You may find it helpful to keep the following guidelines in mind when creating equivalence classes:

* If an input condition specifies a range, at least one valid (the set of values in the range) and two invalid equivalence sets (the set of values less than the lowest extreme of the range, and the set of values greater than the largest extreme) are defined.
* If an input condition specifies a set’s member, at least one valid (the set itself) and one invalid equivalence set (the valid set’s complement) are defined
* If an input condition requires a specific value, then one valid (the set containing the value itself) and two invalid equivalence sets (the set of values less than and the set greater than the value) are defined.

Each equivalence set should be recorded under the appropriate input.

• Qa.1. Do you have enough information to construct the equivalence sets for each input? Can you specify the boundaries of the sets at an appropriate level of detail?

• Qa.2. According to the information in the requirements, are the sets constructed so that no value appears in more than one set?

• Qa.3. Do the requirements state that a particular value should appear in more than one equivalence set? (That is, do they specify more than one type of response for the same value?) Do the requirements specify that a value should appear in the wrong equivalence set?

***Part b: Testing equivalence sets.***

For each equivalence set, write test cases, and record them with the associated equivalence set. Select typical test cases as well as values at and near the boundaries of the sets. For example, if the requirement expects input values in the 0 to 100 range, the test cases selected might be 0, 1, 56, 99, and 100. Finally, for each equivalence set, record the expected resulting behavior. (That is, how do you expect the system to respond to the test cases you just made up?)

• Qb.1. Do you have enough information to create test cases for each equivalence set?

• Qb.2. Are there other interpretations of this requirement that the implementer might make on the basis of the description

given? Will this affect the tests you generate?

• Qb.3. Is there another requirement for which you would generate a similar test case but would get a contradictory result?

• Qb.4. Can you be sure that the tests generated will yield the correct values in the correct units? Is the resulting behavior specified appropriately?

1. **PBR Procedure from the User’s Perspective**

User requirements reflect the specific needs or expectations of the software's customers. Organizations sometimes incorporate these requirements into a BRD, but an application that poses extensive user functionality or complex UI issues might justify a separate document specific to the needs of the intended user. User requirements -- much like user stories -- highlight the ways in which customers interact with software.

There is no universally accepted standard for user requirements statements, but here's one common format:

"The [user type] shall [interact with the software] in order to [meet a business goal or achieve a result]." A user requirement in that mold for the industrial laser marking software example looks like:

***"The production floor manager shall be able to upload new marking files as needed in order to maintain a current and complete library of laser marking images for production use."***

General questions: Read each requirement once and record the number and page along with the inputs to the requirement.

* Q1. According to your actual needs, do the requirements in the SRS meet your expectations for the application?
* Q2. Can you identify all the information required for the input and output of the requirement? Do these inputs and outputs meet your expectations?
* Q3. Are there any necessary functions missing?
* Q4. Are functions that are not required specified?
* Q5. Are some requirements descriptions difficult to understand?
* Q5. Are there any items that have not been entered?
* Q6. Are you increasing unnecessary interactions between users? (Are confirmations and approvals that can be done in one time be divided into multiple parts?)
* Q7. Are performance requirements entered?

1. **PBR Procedure from the Designer’s Perspective**

Designers are the core staff of SRS creation. The software requirements specification should be complete, consistent, accurate, and unambiguous, and at the same time, it should be concise, easy to understand, and easy to modify. Since the software requirements specification must be approved by both developers and users, users must be able to understand and find and point out errors, which plays a great role in ensuring the quality of the software system.

Q1: Does each requirement clearly describe the functions to be implemented so that developers have all the necessary information to design and implement those functions?

Do not omit any necessary requirement information, that is, all functions, performances, design constraints of the target software, and expected behavior under all possible situations, are fully reflected in the requirement specification.

Q2: Are the descriptions of functions, performance, etc. in the requirements specification consistent with the user's expectations for the software?

Q3: Can each requirement be implemented within the capabilities and constraints of the known system and environment?

Q4: Are the parts of the requirements specification contradicting each other?

For all requirement descriptions, readers can only have a clear and unified explanation. Since natural language can easily lead to ambiguity, try to express each requirement in a concise and user-friendly language.

Q5: Whether any of the requirements in the requirements specification can be verified and confirmed by technically and economically feasible means?

Q6: Can the format and organization of the requirements specification ensure that additions, deletions, and modifications are relatively easy, and that the revised requirements specification can maintain other attributes softly and well?

Q7: Is it possible to establish a chain of links between each software requirement and its roots and design elements, source code, test cases, link each requirement to the user's original requirements, and reference these requirements for subsequent development and other documentation Provide convenience?

Q8: Are the parts under consideration (under adjustment) clearly stated?