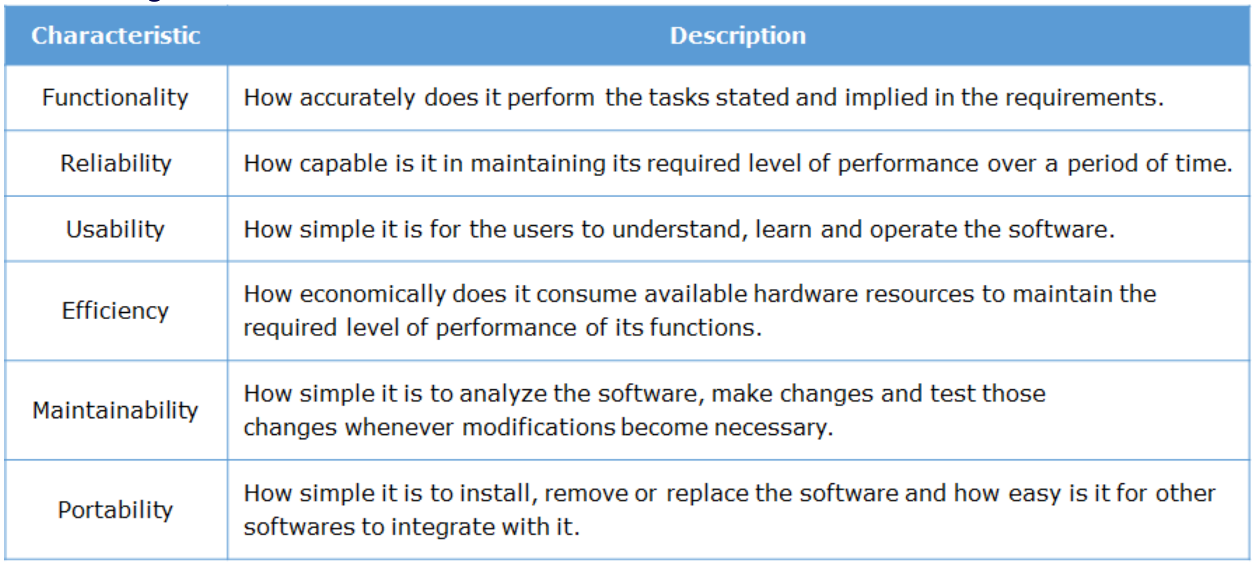
Definition of Software Quality

Software quality is the degree to which a process, component or system conforms to the specified requirements and user expectations.

Characteristics of Software Quality

As per ISO standard model (ISO/IEC 9126), the quality of a software can be evaluated based on its following characteristics.



Characteristic	Description
Functionality	How accurately does it perform the tasks stated and implied in the requirements.
Reliability	How capable is it in maintaining its required level of performance over a period of time.
Usability	How simple it is for the users to understand, learn and operate the software.
Efficiency	How economically does it consume available hardware resources to maintain the required level of performance of its functions.
Maintainability	How simple it is to analyze the software, make changes and test those changes whenever modifications become necessary.
Portability	How simple it is to install, remove or replace the software and how easy is it for other softwares to integrate with it.

The organization creating the software is responsible for ensuring its quality. It is done by conducting tests on the software and the processes used to build the software.

Software Testing and Defects

Software Testing

Software testing is an activity (or set of activities) performed to compare the expected outcome with the actual outcome of one specific task in the system, component or process and report the defects found in the system.

Defect

A defect is a fault in the system, component or process due to which the expected outcome and the actual outcome of a test activity do not match.
The other terms used to refer to a defect are bug, error, anomaly, etc.

Software Testing is the generic term applied to all the activities that are performed to ensure that the quality characteristics of the software meet the required/expected standards.

The objective of software testing is to eliminate defects in a software by

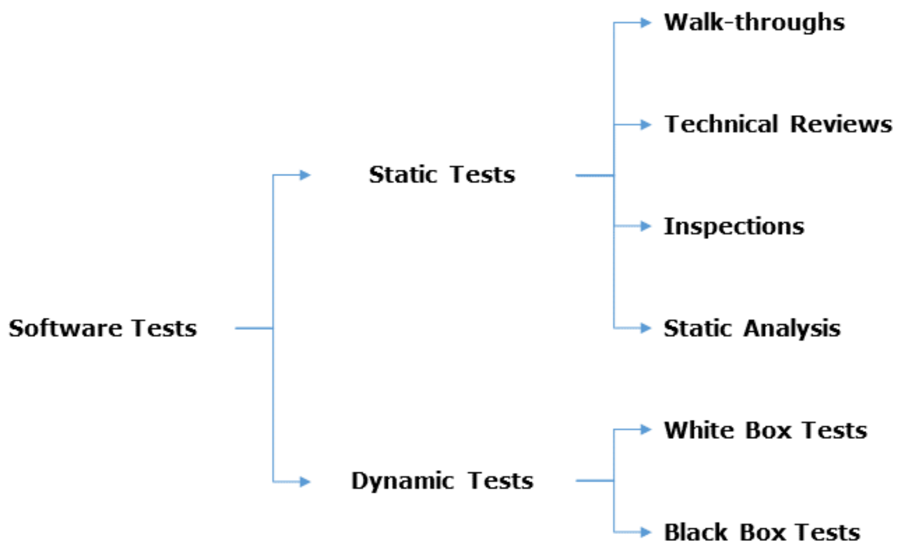
1. Preventing defects from entering the software
2. Identifying and reporting the defects existing in the software

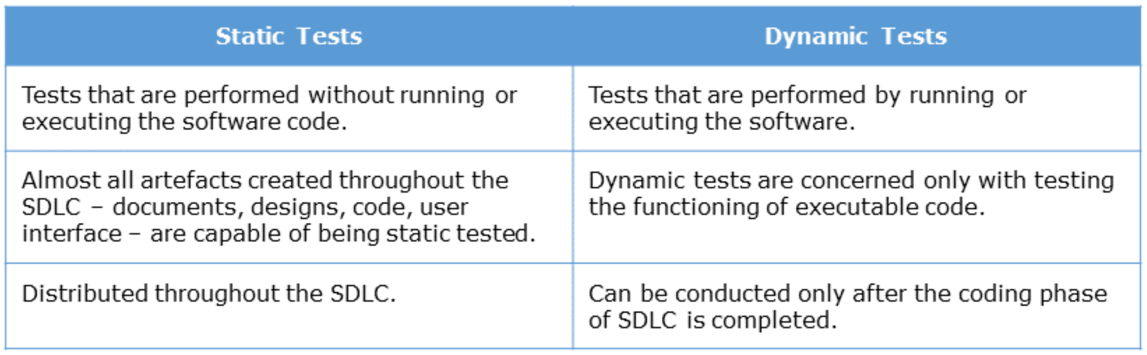
Types of Software Tests

At a very high level tests are categorized as

- **Formal tests** : Tests that document their expected results, actual results and defects found.
- **Informal tests** : Tests that are not documented completely.

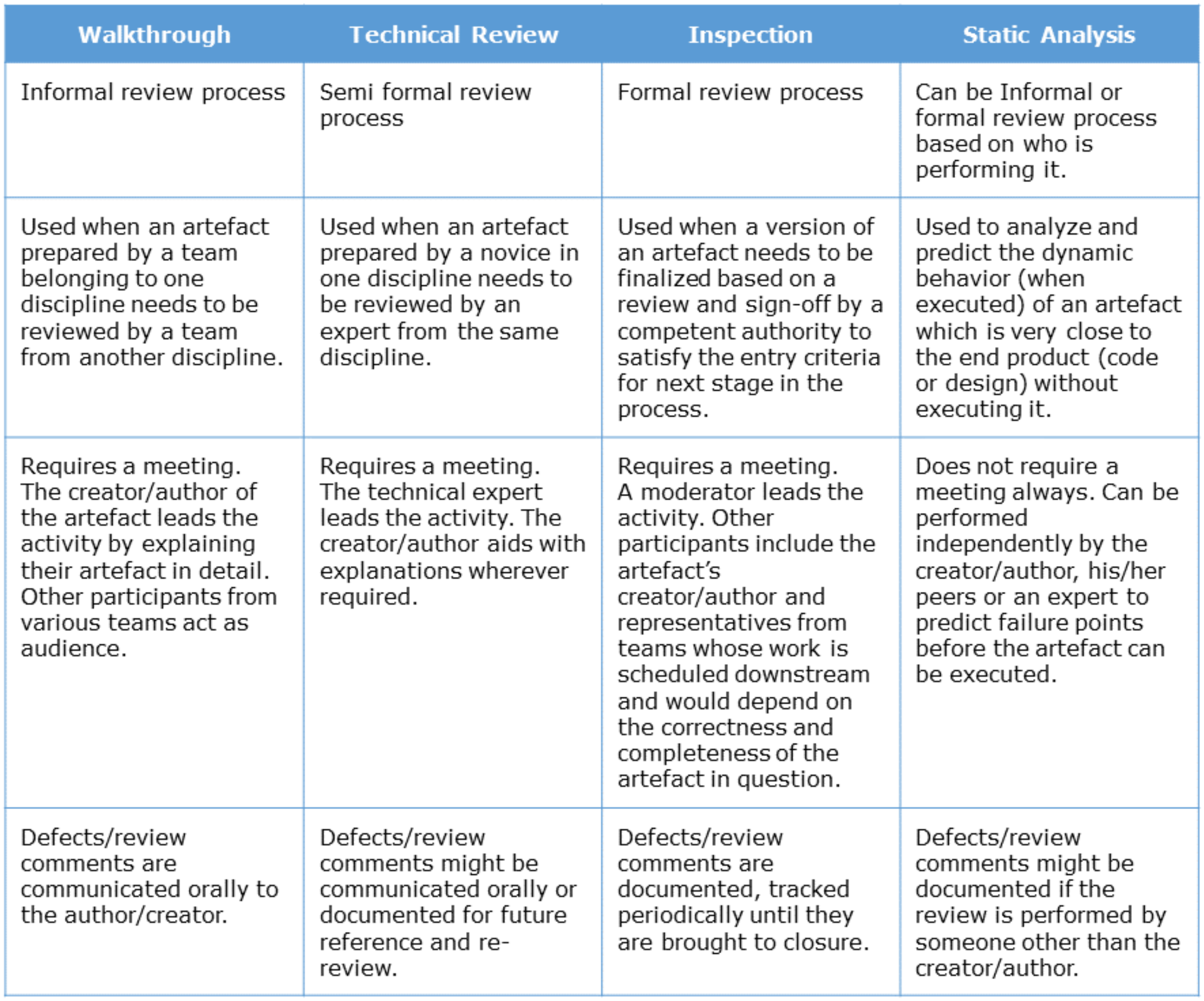
Given below is a more scientific classification of tests, based on the techniques employed to conduct them.





Static Tests	Dynamic Tests
Tests that are performed without running or executing the software code.	Tests that are performed by running or executing the software.
Almost all artefacts created throughout the SDLC – documents, designs, code, user interface – are capable of being static tested.	Dynamic tests are concerned only with testing the functioning of executable code.
Distributed throughout the SDLC.	Can be conducted only after the coding phase of SDLC is completed.

Types of Static Tests



Walkthrough	Technical Review	Inspection	Static Analysis
Informal review process	Semi formal review process	Formal review process	Can be Informal or formal review process based on who is performing it.
Used when an artefact prepared by a team belonging to one discipline needs to be reviewed by a team from another discipline.	Used when an artefact prepared by a novice in one discipline needs to be reviewed by an expert from the same discipline.	Used when a version of an artefact needs to be finalized based on a review and sign-off by a competent authority to satisfy the entry criteria for next stage in the process.	Used to analyze and predict the dynamic behavior (when executed) of an artefact which is very close to the end product (code or design) without executing it.
Requires a meeting. The creator/author of the artefact leads the activity by explaining their artefact in detail. Other participants from various teams act as audience.	Requires a meeting. The technical expert leads the activity. The creator/author aids with explanations wherever required.	Requires a meeting. A moderator leads the activity. Other participants include the artefact's creator/author and representatives from teams whose work is scheduled downstream and would depend on the correctness and completeness of the artefact in question.	Does not require a meeting always. Can be performed independently by the creator/author, his/her peers or an expert to predict failure points before the artefact can be executed.
Defects/review comments are communicated orally to the author/creator.	Defects/review comments might be communicated orally or documented for future reference and re-review.	Defects/review comments are documented, tracked periodically until they are brought to closure.	Defects/review comments might be documented if the review is performed by someone other than the creator/author.

Once the coding is completed, dynamic tests are carried out on the software. These tests are classified further into two categories.

Black Box Testing	White Box Testing
The software is validated by the appearance and behavior and outputs of its User Interface (UI). There is no visibility into the actual underlying code.	The software is validated by the functionality of its program modules by reviewing the code or monitoring the state of the software and its resources at different points in the program flow during execution. There is complete access and visibility into the code.
Validations are done from the software users' perspective (for compliance with the requirements.)	Validations are done from the programmers' perspective (for compliance with the programming standards, best practices and design.)
The objective is to identify the presence of defects in the underlying code. (not locating it)	The objectives can be • To predict UI behavior • To locate a defect • To validate a fixed defect • To check for compliance with design and programming standards.

Quality Assurance and Quality Control

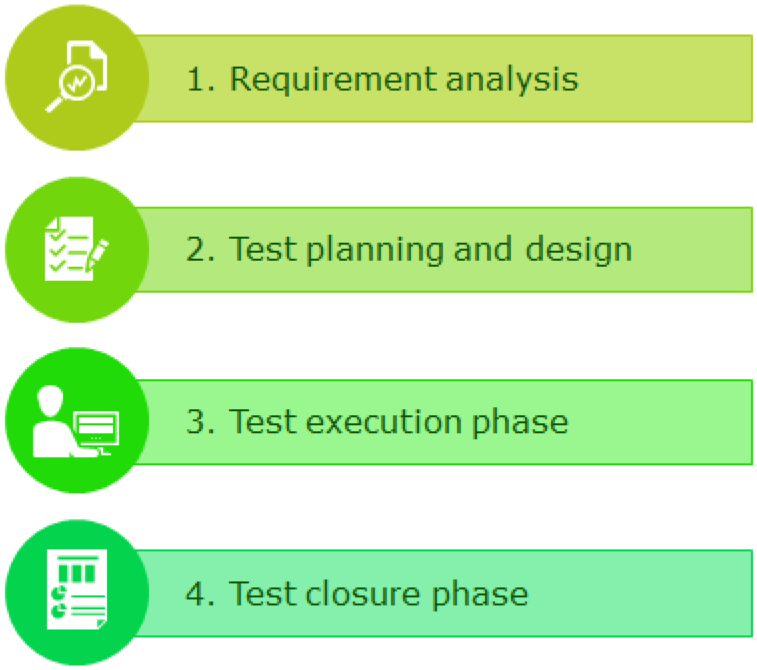
All the software testing activities, conducted by individuals/teams/organizations building a software, are classified into two categories.

Software Quality Assurance (SQA)	Software Quality Control (SQC)
Activities conducted to ensure that processes involved in building the software are correct.	Activities conducted to ensure that the software is built correct. (as per the requirements)
The objective of SQA is to prevent the defects from being injected into the software.	The objective of SQC is to identify the defects that have been injected into the software and fix them.
The entities under test are the processes used for building the software.	The entity under test is the software itself.
SQA activities are conducted during all the phases of the Software Development Life Cycle (SDLC)	SQC activities are generally conducted during the 'Testing' phase of the SDLC.
Tests conducted as a part of SQA activities are termed **Verifications**. They help to answer the question "Are we building the **product right**?"	Tests conducted as a part of SQC activities are termed **Validations**. They help to answer the question "Are we building the **right product**?"
The responsibility of SQA is distributed across all stakeholders of the SDLC – users/clients, business analysts, developers and testers.	The responsibility of SQC generally lies with the testing team (for identifying defects) and development team (for fixing defects)
SQA activities majorly comprise of static tests.	SQC activities majorly comprise of dynamic tests.
SQA documentation is done mostly using checklists, analytical reports and status reports.	SQC documentation consists proof of testing and details of defects.

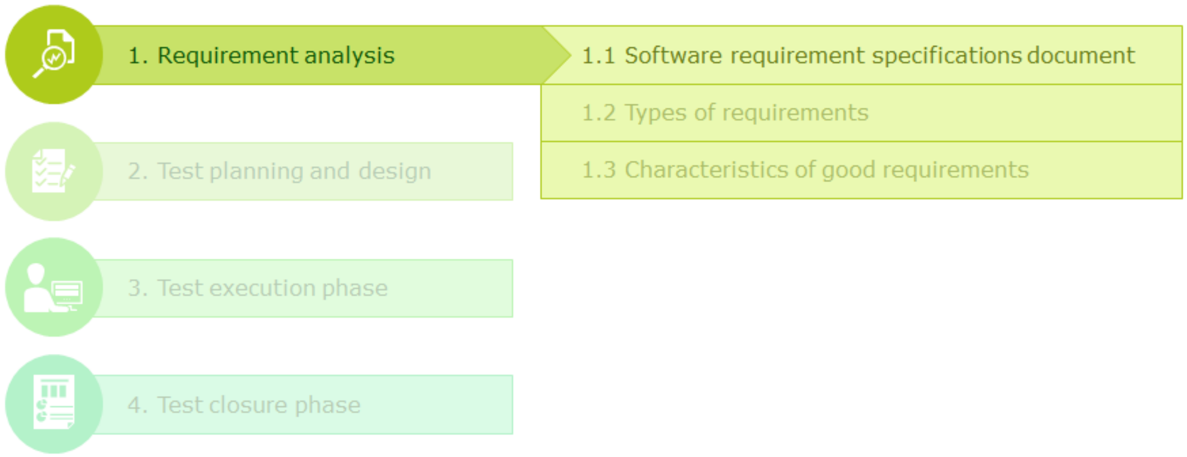
Software Test Life Cycle

If you are a part of the testing team, you will be involved in the SQA and SQC activities associated with the System Testing stage of SDLC.

The development team works based on the SDLC to build a software. Similarly, the testing team works based on the Software Test Life Cycle (STLC) to test a software. The STLC consists of four distinct stages which are sequential.



1.1 Software Requirement Specification Document


1. Requirement analysis
2. Test planning and design
3. Test execution phase
4. Test closure phase
1.1 Software requirement specifications document
1.2 Types of requirements
1.3 Characteristics of good requirements

Requirements for the software to be built, are collected from the end users/clients by analysts and are documented in the form of Software Requirements Specifications (SRS).

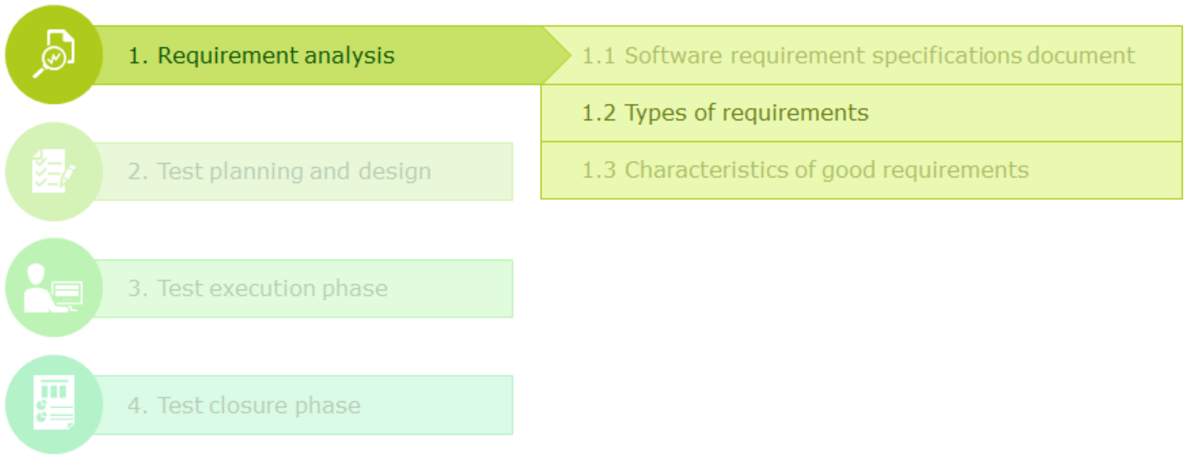
Definition

A Software Requirements Specification is a documented definition of a condition or ability that needs to be possessed by a system or its component to attain an objective of the user.

Significance

Requirement analysis is the first and the most significant stage in the SDLC as well as STLC because:

- Requirements are the basis to develop and validate a software.
- It provides the basis for planning the project, estimating the cost and predicting risks, if any.
- An incorrect or missing requirement will result in a bad or unusable software, no matter how perfectly all the other stages of the SDLC and STLC are executed.
- If requirements are reviewed carefully, omissions, misunderstandings and inconsistencies can be revealed in the initial part of the development cycle when they are easier to rectify. Thus it would save the time and effort spent on redesign, recoding, and retesting.


1. Requirement analysis
1.1 Software requirement specifications document
1.2 Types of requirements
1.3 Characteristics of good requirements
2. Test planning and design
3. Test execution phase
4. Test closure phase

Understanding the types of requirements is important because, depending on the requirement type, the test environment characteristics and test strategies might differ.

Broad categories of requirements are:

1. Functional Requirements

A functional requirement defines the function(s) of a software system or its component. A function is defined in terms of the required output data/behavior of a software (or its component) for a specific combination of input data and/or actions.

In general, functional requirements state WHAT the system should do.

Functional requirements are generally the first to be validated.

Examples:

1. The default value of the gender field must be <Blank>. The system shall allow the user to select either 'Male' or 'Female' from a drop-down menu.
2. If the gender field is set to <Blank> and the user clicks on the 'Submit' button, the system shall display the error message "Please select a valid gender" on the top of the page.

2. Non-Functional Requirements

Non-functional requirements define the constraints under which functional requirements shall operate. They are also called as 'quality attributes' of the system.

In general, non-functional requirements state HOW the system should be.

Some of the major types of non-functional requirements are:

1. **Hardware constraints**: Minimum hardware, database, connectivity configurations under which the software should be able to function. This will give insights into the required configuration for the test environments.
2. **Performance**: Acceptable response times of each component in different possible situations. These requirements should be tested in an environment (hardware configuration) as close as possible to the real-time environment and will require specialized technical knowledge on hardware configurations and performance testing software.
3. **Usability**: UI characteristics, ease of use for a specific user group (E.g., accessibility characteristics for differently-abled users), etc. Testing these requirements requires testers with specialized skill-sets on design.
4. **Security**: These requirements define the access levels to different categories of information present in the system; authentication methodologies, encryption standards for personal data, payment information, etc. Testing these requirements requires specialized skill set in terms of knowledge of security protocols and hacking methods.
5. **Compliance**: Compliance to internal design standards(E.g., branding), government and legal standards(disclaimers, warnings, informational elements, information collected, etc.), accessibility standards for use by differently abled users. These requirements might be tested and certified by third party organizations or independent test groups.

Examples:

1. In case the gender field is set to <Blank> and the user clicks on the 'Submit' button, the error message should be displayed within 5 seconds.
2. The font used for the error message text should be Calibri. The font size of the error message text should be 12. The color used for the error message text should be red.

Problem Statement:

Objective

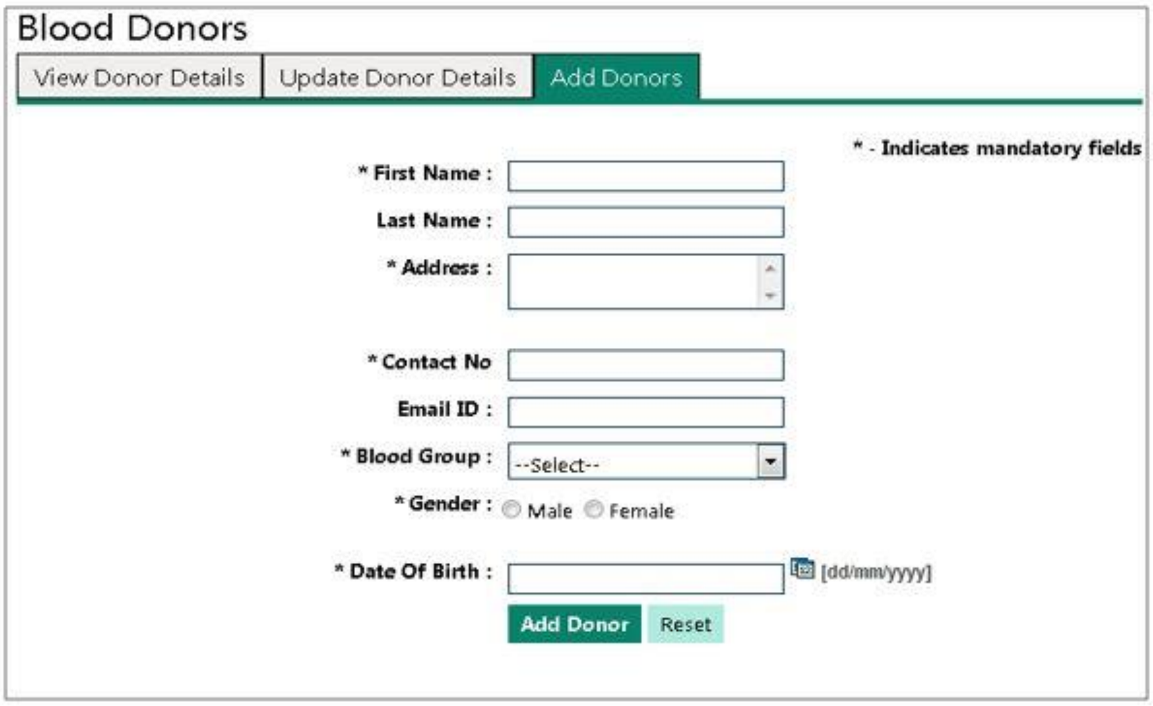
To understand the difference between functional and non-functional requirement.

Problem Statement

Health Management System (HMS) is a hospital management software. Below are the requirements for a new 'Blood Bank Management' portal that needs to be added to the software. Classify the given requirement statements for the 'Blood Bank Management' portal into functional and non-functional requirements.

1. In HMS, there should be a Blood Bank Management Portal where an Admin can Add, View and Update Donor details.
2. In Add donor details module, the user needs to enter first name, address, contact no, blood group, gender and date of birth to successfully add the donor details in the database.
3. User can also provide last name and email id.
4. User should not be able to enter any special character in contact no. The data entered should be restricted to 12 digits.
5. User should enter a valid email id.
6. A list of various blood groups should be populated. User should be restricted to select blood group from this list.
7. The application should support 10,000 concurrent users.
8. User should choose gender from the data displayed on the application page. Male and female should be displayed as data.
9. Application provides user the flexibility to enter the date of birth or select a date from the calendar provided. Donors who are above 18 years of age are eligible to register in the HMS application.
10. The users must be capable of accessing the application 24 X 7.
11. User should be able to reset the fields on the page.
12. Success message “Data added successfully” should be displayed on the same page after successful submission of valid data.
13. The application pages should respond to user requests within 7 seconds.

Note: The application page should look like the one below.



Blood Donors

View Donor Details | Update Donor Details | Add Donors

* - Indicates mandatory fields

* First Name :	
Last Name :	
* Address :	
* Contact No	
Email ID :	
* Blood Group :	--Select--
* Gender :	○ Male ○ Female
* Date Of Birth :	[dd/mm/yyyy]

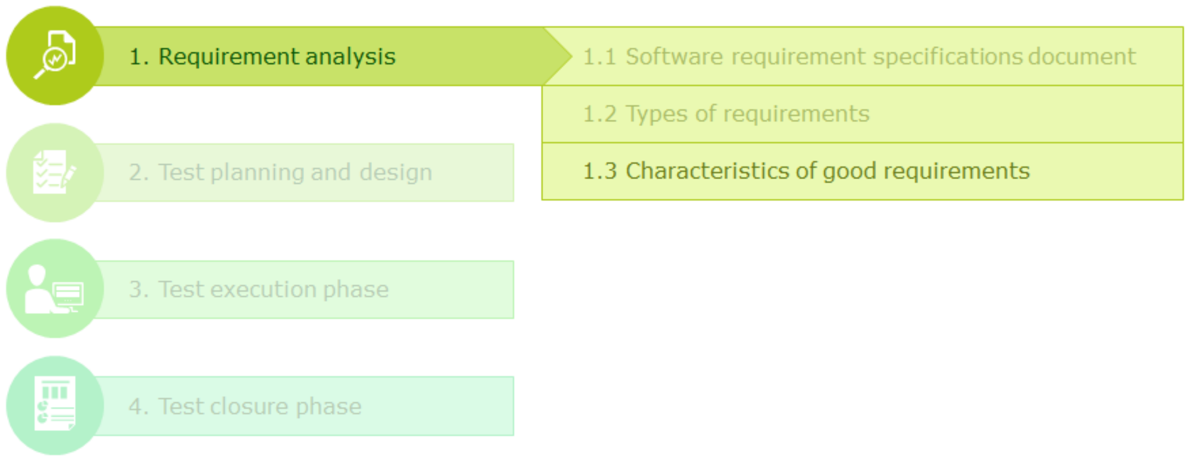
Solution to Exercise 1 – Requirements analysis

Functional Requirements:

Requirement numbers 1,2,3,4,5,6,8,9,11 and 12

Non-Functional Requirements:

Requirement numbers 7, 10 and 13


1. Requirement analysis
1.1 Software requirement specifications document
1.2 Types of requirements
1.3 Characteristics of good requirements
2. Test planning and design
3. Test execution phase
4. Test closure phase

Requirement review is one of the important tasks of the testing team. Finding and fixing gaps in the SRS document ensures smooth operation of the remaining stages of the STLC.

Requirement review meetings happen in the Requirement Analysis stage, in the presence of all stakeholders of the project - Users/Clients, Business Analysts, System Analysts, Developers and Testers.

From a testing team's perspective, below are the important requirement characteristics that you should be looking for while analyzing and reviewing the SRS document.

Completeness

An SRS document is considered to be complete if

- All the user's/customer's expectations documented in higher level requirements, like Business Requirement Specification (BRS) document, have been addressed in the SRS document and nothing has been missed out.
- All significant aspects, both functional and non-functional, of each requirement specification has been addressed.
- It states the expected responses of the software (and its components) to all possible classes of input data, both valid and invalid, in all possible situations.
- Definition of all business-specific/software-specific abbreviations in its full form and their meanings.
- There are no parts/sections in the SRS document marked as To Be Determined (TBD). If any part is marked so, it should also contain information on cause of unavailability of information, expected timeline for availability of the required information and the person responsible for making the information available in the SRS document.

Example: The requirement statement "*Allow the user to login to the system only if the user id and password combination is authenticated by the security module.*" is an incomplete requirement because it does not tell what the system is supposed to do if either the user id or password is invalid.

Unambiguity

A requirement specification said to be unambiguous if each stated requirement has only one possible interpretation to all the project stakeholders- users, developers and testers.

Example: The requirement statement "*The User_ID field shall not accept input longer than 25 characters*" is ambiguous because it can be interpreted in any of the following possible ways

- Throw an error message whenever the user enters more than 25 characters in the User_ID field.
- Do not respond for any inputs from the keyboard after the user has typed 25 characters in the User_ID field.
- Truncate the contents of the User_ID field to 25 characters once the user moves the cursor out of that field.

Whenever a word used in a specific context might have several meanings, it must be included in a glossary where its meaning is more precise.

Example: The terms like '*Acceptable Response Time*' must be made more specific in terms of measurable units like 3 seconds.

Consistency

A software requirement is said to inconsistent if it does not agree with or conflicts with

- Any higher level requirements like the BRS document.
- Another software requirement in the same SRS document.
- Inputs and outputs of existing software and their interfaces with which the new software or it's components need to communicate.

Example: The following requirement statements are in conflict with each other.

"*Every page in the software application must have a 'Home' link at the bottom of the page and anytime the user clicks it, immaterial of the page or the state of activity in that page, the software should navigate to the home page*"

"*If an error message is displayed in page 4, the user should not be allowed to navigate to any page until all the errors for the corresponding page are fixed*"

Testability

A requirement specification is testable if and only if there exists some finite cost-effective process with which a tester can check that the software product meets that requirement.

A requirement cannot be tested by the tester if he/she does not have access to the required hardware configuration or software configuration due to cost and time factors.

Also as a general thumb rule,

- All ambiguous requirements are not testable as there is no single outcome that can be called completely correct or completely wrong.
- All inconsistent requirements are not testable as it is impossible to prove that the system is either working or not working as per the requirement.

Example: The requirement "*The home page's loading time should not drop below 5 seconds as long as the number of simultaneous users does not exceed 1 million users*" cannot be tested unless the tester has a software tool which can simulate 1 million simultaneous users or 1 million real testers with individual machines to access the home page simultaneously.

Requirement Characteristics - Exercise

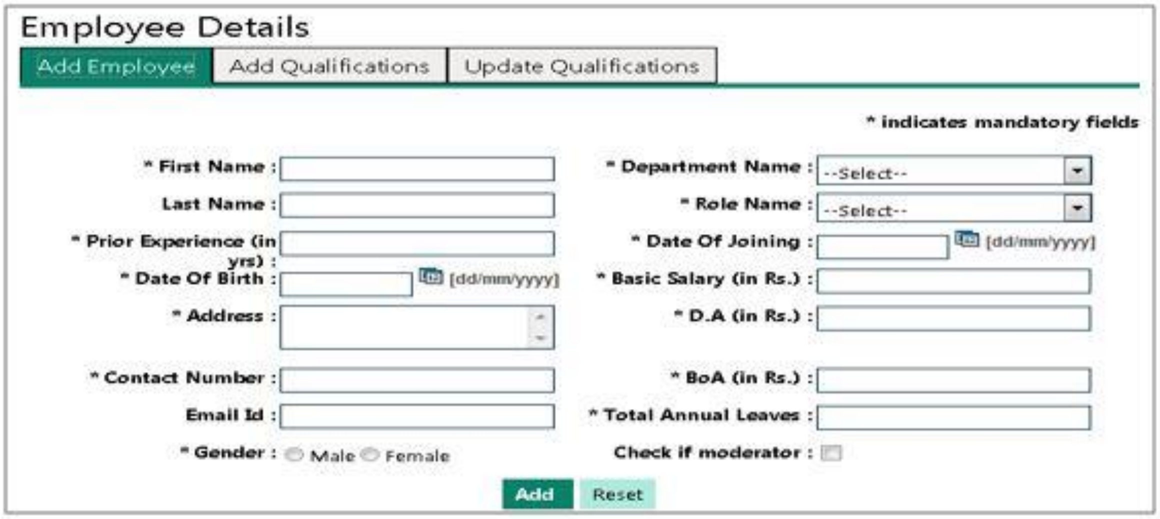
Problem Statement:

Objective

To understand requirement characteristics by reviewing and classifying the requirement statements as good or bad.

Problem Statement

Read through some of the requirements specified for the 'Add Employee' module of Health Management System that needs to be built as per the figure given below and classify them as good or bad depending on the various characteristics of software requirements.


Employee Details

* indicates mandatory fields

* First Name :		* Department Name :	--Select--
Last Name :		* Role Name :	--Select--
* Prior Experience (in yrs) :		* Date Of Joining :	[dd/mm/yyyy]
* Date Of Birth :	[dd/mm/yyyy]	* Basic Salary (in Rs.) :	
* Address :		* D.A (in Rs.) :	
* Contact Number :		* BoA (in Rs.) :	
Email Id :		* Total Annual Leaves :	
* Gender :	Male Female	Check if moderator :	

Application level Requirements:

1. All the field names marked with an '*' before them in the look up diagram are supposed to be considered as mandatory fields.

2. A pop-up calendar function should be implemented for all date fields, using which the user can click and select the required date value to be entered in the corresponding date field.

3. The page and its fields should be user friendly.

4. The employee details report must be auto generated by the system and sent to the HMS Admin.

Field Level Requirements:

1. There must be a field "First Name" of type textbox. It should not allow the user to enter more than 30 characters.

2. There should be a field "Date of Birth" of type textbox. It should accept only a date value as input and only in the date format *mm/dd/yyyy*. If an invalid date value or an invalid format is entered, then an error message "Invalid Date of Birth" should be displayed.

3. There should be a 'Reset' field of type button. Upon clicking this button

1. All existing values in all the text fields in the page should be cleared and fields should be set to blank.

2. All existing selections in the drop-down fields, radio button fields and check box fields should be discarded and the fields have to go back to their unselected state.

3. All existing values in the date fields must be cleared and the fields should be set to blanks

4. The system must take the first two digits of the ‘Contact Number’ as STD code.

Solution

Solution to Exercise 2 – Requirement Characteristics

Application Level Requirement # 1

This is a bad requirement because it’s incomplete. There is nothing specifically stated about

1. How the ‘mandatory’ ness has to be implemented. There are multiple ways in which this functionality can be implemented like error messages, warning messages, pop up boxes, disabling the ‘Add’ button or a combination of options described.

2. There is no mention about which fields should be mandatory.

Application Level Requirement # 2

This is a good requirement.

Application Level Requirement # 3

This is a bad requirement as ‘User Friendly’ is a very ambiguous term and can mean different things to users, programmers and testers. There is no baseline using which you, as a tester, can validate this requirement.

Application Level Requirement # 4

This is a bad requirement since it is incomplete. It lacks the information like

1. In what frequency the report should be generated

2. The file type of the report

3. The exact content of the report in terms of data from which fields.

4. Using which channel of communication the report should be shared with the HMS Admin.

Field Level Requirement # 1

This is a bad requirement since it is ambiguous. The ambiguity exists on how the system is supposed to work.

The possibilities are:

1. The system shall not let the user enter more than 30 characters
2. The system shall truncate the entered string to 30 characters
3. The system shall display an error message if the user enters more than 30 characters

Field Level Requirement # 2

Though this requirement looks complete, unambiguous and testable, it is in conflict with page level requirement # 2 for the date fields. It will confuse the developer and tester as to whether only a calendar can be used to input date values or they have to be typed as text or both should be allowed. Hence this is an inconsistent requirement.

Field Level Requirement # 3

This is a good requirement.

Field Level Requirement # 4

This is an incomplete requirement as there is no data provided on the type of the field, the size of the field or valid and invalid values for the field.