NeST-GDL-GEN.05
Version 0.2
June, 2016
Status: Draft

Guidelines for
Software Requirements Specification

Government of India
Department of Electronics & Information Technology
Ministry of Communications & Information Technology
New Delhi-110 003

Change History

Version No.	Release Date	Change Details
0.1	April, 2016	First Draft of Guidelines for Software Requirements Specification
0.2	June, 2016	Review comments received from working group incorporated

Metadata of the Standard

S. No.	Data elements	Values
1.	**Title**	Guidelines for Software Requirements Specification
2.	**Title Alternative**	FRS
3.	**Document Identifier**	NeST-GDL-GEN.05
4.	**Document Version, month, year of release**	Version: 0.2 June, 2016
5.	**Present Status** *(Draft/Released/Withdrawn)*	Draft
6.	**Publisher**	Department of Electronics and Information Technology (DeitY), Ministry of Communications & Information Technology (MCIT), Government of India (GoI)
7.	**Date of Publishing**	
8.	**Type of Standard Document** *(Standard/ Policy/ Technical/ Specification/ Best Practice /Guideline / Framework /Procedure)*	Guidelines
9.	**Enforcement Category** *(Mandatory / Recommended)*	Recommended
10.	**Creator** *(An entity primarily responsible for making the resource)*	NeST (STQC)
11.	**Contributor** *(An entity responsible for making contributions to the resource)*	DeitY
12.	**Brief Description**	This guideline document describes the Software Requirements Specification (SRS), preparation of a SRS and provides a SRS template.
13.	**Target Audience** *(Who would be referring / using the Standard)*	eGD (DeitY), NeST(STQC), Working group, Users
14.	**Owner of approved Standard**	DeitY, MCIT, New Delhi
15.	**Subject** (Major Area of Standardization)	General (Software Requirement Specification)
16.	**Subject. Category** (Sub Area within major area)	Software Requirement Specification
17.	**Coverage. Spatial**	India
18.	**Format**	PDF

	(PDF/A at the time of release of final Standard)	
19.	**Language** *(To be translated in other Indian languages later)*	English
20.	**Copyrights**	DeitY, MCIT, New Delhi
21.	**Source** *(Reference to the resource from which present resource is derived)*	Refer to section 4: References
22.	**Relation** *(Relation with other e-Governance standards notified by DeitY)*	Guideline for e-Gov Project Life Cycle

Table of Contents

1. Introduction

An SRS is a document conveying the customer requirements for a system and/or software to the developer's organization prior to the design and development work. SRS assures that both the client and the developer's organization understand the other's requirements from that perspective at a given point in time. The SRS document states in precise and explicit language, the functions and capabilities a system or software system must provide, as well as states any required constraints by which the system must abide.

The SRS is often referred to as the "parent" document because all subsequent project documents, such as design specifications, software architecture specifications, testing and validation plans, and other documentation are related to it.

It is important to note that an SRS contains mainly the functional and non-functional requirements and does not offer design details, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

An SRS minimizes the time and effort required by developers to achieve desired goals and also minimizes the development cost. SRS establishes the basis for an agreement between customers and suppliers on what the software product is to do as well as what it is not expected to do.

A good SRS defines how software will interact with system hardware, other software and human users in real-world situations. Parameters such as performance, security, availability, portability, maintainability, and recovery from adverse events are also defined and evaluated.

2. Purpose

The purpose of this document "Guidelines for Software Requirements Specification (SRS)" is to provide explanation, description to prepare a SRS and explanation to use SRS template.

The SRS document will draw inputs from Request for Proposal (RFP) document or Functional Requirements Specification (FRS) and will provide inputs to development team to design the software.

3. Scope

The scope of this document is to provide a description SRS document for system and software and guidelines to prepare a SRS for e-Government systems.

4. References

- IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications
- IEEE Std 1233, 1998 Edition, IEEE Guide for Developing System Requirements Specification
- IEEE/EIA Std 12207.0-1996, Software life cycle processes

- IEEE/EIA Std 122207.1-1997, Software life cycle processes--Life cycle data
- IEEE/EIA Std 12207.2-1997, Software life cycle processes--Implementation considerations
- ISO/IEC 12207 (IEEE Std 12207-2008): Systems and Software Engineering - Software Life Cycle Processes
- ISO/IEC 15288 (IEEE Std 15288-2008): Systems and Software Engineering - System Life Cycle Processes

5. Abbreviations & Acronyms

FRS: Functional Requirements Specification
RFP: Request For Proposal
SRS: Software Requirements Specification
SyRS: System Requirements Specification
RTM: Requirements Traceability Matrix

6. Terms & Definitions

Software: Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.
Types of software may include application software; support software; system software.

System: A collection of components organized to accomplish a specific function or set of functions.
Also defined as an interdependent group of people, objects, and procedures constituted to achieve defined objectives or some operational role by performing specified functions. A complete system includes all of the associated equipment, facilities, material, computer programs, firmware, technical documentation, services, and personnel required for operations and support to the degree necessary for self-sufficient use in its intended environment.

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 or system component to satisfy a contract, standard, specification, or other formally imposed documents.
(3) A documented representation of a condition or capability as in (1) or (2).
Types of requirements include design requirement; functional requirement; implementation requirement; interface requirement; performance requirement; physical requirement.

Requirements Specification: A document that specifies the requirements for a system or component.
Typically included are functional requirements, performance requirements, interface

requirements, design requirements and development standards.

Functional Requirement: A requirement that specifies a function that a system or system component must be able to perform.

Functional Specification: A document that specifies the functions that a system or component must perform. Often part of a requirements specification.

Request for proposal [Tender]: A document used by the acquirer as the means to announce its intention to potential bidders to acquire a specified system, software product or software service.

Software Requirements Specification: Documentation of the essential requirements (functional, performance, design constraints and attributes) of the software and its external interfaces.

System Requirements Specification (SyRS): A structured collection of information that embodies the requirements of the system.

Traceability:
(1) The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match.
(2) The degree to which each element in a software development product establishes its reason for existing; for example, the degree to which each element in a bubble chart references the requirement that it satisfies.

Traceability Matrix: A matrix that records the relationship between two or more products of the development process; for example, a matrix that records the relationship between the requirements and the design of a given software component.

7. Software Requirements Specification (SRS)

SRS is generally defined as a document that specifies the requirements for a system or software which typically includes functional requirements, performance requirements, interface requirements, design requirements, and development standards.
There are two types of requirements documents; the System Requirements Specification (SyRS) and Software Requirements Specification (SRS).

System Requirements Specification (SyRS):

A SyRS document specifies the system requirement, not the software requirements. It is a system engineering activity; software is always an element of a larger system, e.g., Airplanes, the Space Shuttle, a cell phone. The requirements for the software components of the system are then derived from the system requirements.

The IEEE STD 1233 is a guide for developing System Requirements Specification (SyRS).

Software Requirements Specification (SRS):

An SRS document describes the essential software requirements (functional, performance, design constraints and attributes) of the software and its external interfaces.

An SRS document establishes understanding of the software product is to do, as well as what it is not expected to do (may be accompanied by definition document for clarity). It is written in natural language and supplemented by formal/ semi-formal description to allow for a more precise and concise description of software architecture.

A well-documented SRS provides realistic basis for estimating product costs, risks and schedules. It also permits rigorous assessment of requirements before design.

The key quality indicators of a good SRS are size, readability, specification, depth, and text structure.

The IEEE Std 830 is a guide for developing Software Requirements Specification (SRS).

SRS is a comprehensive description of a software system to be developed. It details out functional and non-functional requirements of the software. The requirements may be described using textual details or may include a set of use cases that describe user interactions that the software must provide.

A well-designed, well-written SRS accomplishes four major goals:

- It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the system & software to address those problems. Therefore, the SRS should be written in natural language in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.
- It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.
- It serves as an input to the design specification. The SRS serves as the parent document to subsequent documents, such as the software design specification and statement of work.

Therefore, the SRS must contain sufficient detail in the functional system requirements so that a design solution can be devised.

- It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

SRS is typically developed during the first stages of "Requirements Development," which is the initial product development phase in which information is gathered about what requirements are needed - and what are not. This information-gathering stage can include onsite visits, questionnaires, surveys, interviews, and perhaps a Return-On-Investment (ROI) analysis or needs analysis of the customer or client's current business environment. Then the actual specification is written after the requirements have been gathered and analyzed.

7.1 SRS Elements:

An SRS document addressed the following elements:

a) Interfaces
b) Functional Capabilities
c) Performance Levels
d) Data Structures/ Elements
e) Safety
f) Reliability
g) Security/ Privacy
h) Quality
i) Constraints and Limitations
j) Assumptions and Dependencies

7.2 SRS Quality Characteristic:

A good SRS is one that fully addresses all the customer requirements for a particular product or system. A quality SRS is one in which the requirements are tightly, unambiguously, and precisely defined in such a way that leaves no other interpretation or meaning to any individual requirement. The quality characteristics of a well document SRS are:

a) Complete
b) Consistent
c) Correct
d) Modifiable
e) Ranked
f) Testable
g) Traceable
h) Unambiguous

i) Valid
j) Verifiable

The quality characteristics are described below:

a) Complete

SRS defines precisely all the go-live situations that will be encountered and the system's capability to successfully address them.

An SRS is complete if, and only if, it includes the following elements:

a) 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.

b) 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.

c) Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure.

Use of TBDs

Any SRS that uses the phrase "To Be Determined" (TBD) is not a complete SRS. The TBD is, however, occasionally necessary and should be accompanied by:

a) A description of the conditions causing the TBD (e.g., why an answer is not known) so that the situation can be resolved;

b) A description of what must be done to eliminate the TBD, who is responsible for its elimination, and by when it must be eliminated.

b) Consistent

SRS capability functions and performance levels are compatible, and the required quality features (security, reliability, etc.) do not negate those capability functions.

Consistency here 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 graphical.

2) One requirement may state that control signal be set "ON" while another may state that control signal be set "OFF".

b) There may be logical or temporal conflict between two specified actions. For example,

1) One requirement may specify that the program will take two inputs and another may specify that the program will take four inputs.

2) One requirement may state that event "A" must always follow event "B" while another may require that events "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.

c) Correct

SRS precisely defines the system's capability in a real-world environment, as well as how it interfaces and interacts with it. 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.

d) Modifiable

The logical, hierarchical structure of the SRS should facilitate any necessary modifications (grouping related issues together and separating them from unrelated issues makes the SRS easier to modify).

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 cross referencing;

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.

e) Ranked

Individual requirements of an SRS are hierarchically arranged according to stability, security, perceived ease/ difficulty of implementation, or other parameter that helps in the design of that and subsequent documents.

An SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement.

Typically, all of the requirements that relate to a software product are not equally important. Some requirements may be essential, especially for life-critical applications, while others may be desirable.

Each requirement in the SRS should be identified to make these differences clear and explicit. Identifying the requirements in the following manner helps:

a) Have customers give more careful consideration to each requirement, which often clarifies any hidden assumptions they may have.

b) Have developers make correct design decisions and devote appropriate levels of effort to the different parts of the software product.

Degree of Stability

One method of identifying requirements uses the dimension of stability. Stability can be expressed in terms of the number of expected changes to any requirement based on experience or knowledge of forthcoming events that affect the organization, functions, and people supported by the software system.

Degree of Necessity

Another way to rank requirements is to distinguish classes of requirements as essential, conditional, and optional.

a) Essential - Implies that the software will not be acceptable unless these requirements are provided in an agreed manner.

b) *Conditional* - Implies that these are requirements that would enhance the software product, but would not make it unacceptable if they are absent.

c) *Optional* - Implies a class of functions that may or may not be worthwhile. This gives the supplier the opportunity to propose something that exceeds the SRS.

f) Testable

An SRS must be stated in such a manner that unambiguous assessment criteria (pass/fail or some quantitative measure) can be derived from the SRS itself.

g) Traceable

Each requirement in an SRS must be uniquely identified to a source (use case, government requirement, industry standard, etc.)

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.

h) Unambiguous

SRS must contain requirements statements that can be interpreted in one way only. This is another area that creates significant problems for SRS development because of the use of natural language.

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.

Recommendations to avoid ambiguity are:

Natural Language Pitfalls:

Requirements are often written in natural language (e.g., English). Natural language is inherently ambiguous. A natural language SRS should be reviewed by an independent party to identify ambiguous use of language so that it can be corrected.

Requirements Specification Languages:

One way to avoid the ambiguity inherent in natural language is to write the SRS in a particular

requirements specification language. Its language processors automatically detect many lexical, syntactic, and semantic errors.

One disadvantage in the use of such languages is the length of time required to learn them. Also, many nontechnical users find them unintelligible. Moreover, these languages tend to be better at expressing certain types of requirements and addressing certain types of systems. Thus, they may influence the requirements in subtle ways.

Representation Tools;

In general, requirements methods and languages and the tools that support them fall into three general categories Object, Process, and Behavioral.

Object-oriented approaches organize the requirements in terms of real-world objects, their attributes, and the services performed by those objects.

Process-based approaches organize the requirements into hierarchies of functions that communicate via data flows.

Behavioral approaches describe external behavior of the system in terms of some abstract notion (such as predicate calculus), mathematical functions, or state machines.

The degree to which such tools and methods may be useful in preparing an SRS depends upon the size and complexity of the program. No attempt is made here to describe or endorse any particular tool.

When using any of these approaches it is best to retain the natural language descriptions. That way, customers unfamiliar with the notations can still understand the SRS.

i) Valid

A valid SRS is one in which all parties and project participants can understand, analyze, accept, or approve it. This is one of the main reasons SRSs are written using natural language.

j) Verifiable

A verifiable SRS is consistent from one level of abstraction to another. Most attributes of a specification are subjective and a conclusive assessment of quality requires a technical review by domain experts. Using indicators of strength and weakness provide some evidence that preferred attributes are or are not present.

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.

Nonverifiable 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 nonverifiable because the testing of this quality is theoretically

impossible.

An example of a verifiable statement is

Output of the program shall be produced within 20 sec. of event x 60% of the time; and shall be produced within 30 sec of event x 100% of the time.

This statement can be verified because it uses concrete terms and measurable quantities.

If a method cannot be devised to determine whether the software meets a particular requirement, then that requirement should be removed or revised.

7.3 Quality Measures related to Individual SRS Statements:

1. Imperatives: Words and phrases that command the presence of some feature, function, or deliverable. They are listed below in decreasing order of strength.

Word & Phrase	Description
Shall	Used to dictate the provision of a functional capability.
Must or must not	Most often used to establish performance requirement or constraints.
Is required to	Used as an imperative in SRS statements when written in passive voice.
Are applicable	Used to include, by reference, standards, or other documentation as an addition to the requirement being specified.
Responsible for	Used as an imperative in SRSs that is written for systems with pre-defined architectures.
Will	Used to cite things that the operational or development environment is to provide to the capability being specified. For example, The vehicle's exhaust system will power the ABC widget.
Should	Not used often as an imperative in SRS statements; however, when used, the SRS statement always reads weak. Avoid using Should in your SRSs.

2. Continuances: Phrases that follow an imperative and introduce the specification of requirements at a lower level. There is a correlation with the frequency of use of *continuances* and SRS organization and structure, up to a point. Excessive use of *continuances* often indicates a very complex, detailed SRS. The *continuances* below are listed in decreasing order of use within SRS. Use *continuances* in your SRS, but balance the frequency with the appropriate level of detail called for in the SRS.

a) Below:
b) As follows:
c) Following:
d) Listed:
e) In particular:
f) Support:

3. Directives: Categories of words and phrases that indicate illustrative information within the SRS. A high ratio of total number of *directives* to total text line count appears to correlate with how precisely requirements are specified within the SRS. The *directives* below are listed in decreasing order of occurrence within SRS. Incorporate the use of *directives* in the SRS.

a) Figure
b) Table
c) For example
d) Note

4. Options: A category of words that provide latitude in satisfying the SRS statements that contain them. This category of words loosens the SRS, reduces the client's control over the final product, and allows for possible cost and schedule risks. You should avoid using them in your SRS. The *options* below are listed in the order they are found most often in the SRS.

a) Can
b) May
c) Optionally

5. Weak phrases: A category of clauses that can create uncertainty and multiple/ subjective interpretation. The total number of *weak phrases* found in an SRS indicates the relative ambiguity and incompleteness of an SRS. The *weak phrases* below are listed alphabetically.

Adequate	Be Able To	Easy	Provide For
As A Minimum	Be Capable Of	Effective	Timely
As Applicable	But Not Limited To	If Possible	TBD
As Appropriate	Capability Of	If Practical	
At A Minimum	Capability To	Normal	

6. Size: Used to indicate the *size* of the SRS document, and is the total number of the following:

a) Lines of text
b) Number of imperatives
c) Subjects of SRS statements
d) Paragraphs

7. Text Structure: Related to the number of statement identifiers found at each hierarchical level of the SRS and indicate the document's organization, consistency, and level of detail. The most detailed SRS can go up to nine levels deep. High-level SRS are rarely more than four levels deep. SRS deemed well organized and a consistent level of detail had *text structures* resembling pyramids (few level 1 headings but each lower level having more numbered statements than

the level above it). Hour-glass-shaped *text structures* (many level 1 headings, few a mid-levels, and many at lower levels) usually contain a greater amount of introductory and administrative information. Diamond-shaped *text structures* (pyramid shape followed by decreasing statement counts at levels below the pyramid) indicated that subjects introduced at higher levels were addressed at various levels of detail.

8. Specification Depth: The number of imperatives found at each of the SRS levels of text structure. These numbers include the count of lower level list items that are introduced at a higher level by an imperative and followed by a continuance. The numbers provide some insight into how much of the Requirements document was included in the SRS, and can indicate how concise the SRS is in specifying the requirements.

9. Readability Statistics: Measurements of how easily one can read and understand the requirements document. Four readability statistics are used (calculated by Microsoft Word). While readability statistics provide a relative quantitative measure, don't sacrifice sufficient technical depth in SRS for a number.

a) Flesch Reading Ease index
b) Flesch-Kincaid Grade Level index
c) Coleman-Liau Grade Level index
d) Bormuth Grade Level index

These measures are explained below:

a) The Flesch Reading Ease Readability Formula:

Flesch Reading Ease Formula is considered as one of the oldest and most accurate readability formulas. Rudolph Flesch, an author, writing consultant, and a supporter of the Plain English Movement, developed this formula in 1948. In his article, *A New Readability Yardstick*, published in the Journal of Applied Psychology in 1948, Flesch proposed the Flesch Reading Ease Readability Formula.

To grade a text **Flesch Reading Ease Calculator** is available at:
http://www.readabilityformulas.com/free-readability-formula-tests.php

The Flesch Reading Ease Formula is a simple approach to assess the grade-level of the reader. It has since become a standard readability formula used by many US Government Agencies, including the US Department of Defense. However, primarily, the formula is used to assess the difficulty of a reading passage written in English.

The Flesch Reading Ease Readability Formula

The specific mathematical formula is:

RE = 206.835 – (1.015 x ASL) – (84.6 x ASW)

RE = Readability Ease

ASL = Average Sentence Length (i.e., the number of words divided by the number of sentences)

ASW = Average number of syllables per word (i.e., the number of syllables divided by the number of words)

The output, i.e., RE is a number ranging from 0 to 100. The higher the number, the easier the text is to read.

- Scores between 90.0 and 100.0 are considered easily understandable by an average 5th grader.
- Scores between 60.0 and 70.0 are considered easily understood by 8th and 9th graders.
- Scores between 0.0 and 30.0 are considered easily understood by college graduates.

To draw a conclusion from the Flesch Reading Ease Formula, the best text should contain shorter sentences and words. The score between 60 and 70 is largely considered acceptable. The following table is also helpful to assess the ease of readability in a document:

Score	Readability
90-100	Very Easy
80-89	Easy
70-79	Fairly Easy
60-69	Standard
50-59	Fairly Difficult
30-49	Difficult
0-29	Very Confusing

Though simple it might seem, the Flesch Reading Ease Formula has certain ambiguities. For instance, periods, explanation points, colons and semicolons serve as sentence delimiters; each group of continuous non-blank characters with beginning and ending punctuation removed counts as a word; each vowel in a word is considered one syllable subject to:

(a) -es, -ed and -e (except -le) endings are ignored;

(b) Words of three letters or shorter count as single syllables; and

(c) Consecutive vowels count as one syllable.

b) The Flesch-Kincaid Grade Level Readability Formula

Flesch Grade Level Readability Formula improves upon the Flesch Reading Ease Readability Formula. Rudolph Flesch is the co-author of this formula along with John P. Kincaid. That's why it is also called Flesch-Kincaid Grade Level Readability Test. In his article, A New Readability

Yardstick, published in the Journal of Applied Psychology in 1948, Flesch proposed the Reading Ease Readability Formula.
To grade a text **Flesch Grade Level Calculator** is available at:
http://www.readabilityformulas.com/free-readability-formula-tests.php

In 1976 the US Navy modified the Reading Ease formula to produce a grade-level score by applying the Flesch Grade-Scale formula, or the Kincaid formula. John P. Kincaid was assisted by Fishburne, Rogers, and Chissom, in his research. This formula is known by different names, like Flesch-Kincaid Index, Flesch-Kincaid Grade Level Score, Flesch-Kincaid Scale, Flesch-Kincaid Score, Flesch-Kincaid Readability Score, Flesch-Kincaid Readability Statistics, Flesch-Kincaid Grade Level Index, Flesch-Kincaid Readability Index, Flesch-Kincaid readability equation, and so on. Originally formulated for US Navy purposes, this Formula is best suited in the field of education.

The Flesch-Kincaid Grade Level Readability Formula
Step 1: Calculate the average number of words used per sentence.
Step 2: Calculate the average number of syllables per word.
Step 3: Multiply the average number of words by 0.39 and add it to the average number of syllables per word multiplied by 11.8.
Step 4: Subtract 15.59 from the result.
The specific mathematical formula is:
FKRA = (0.39 x ASL) + (11.8 x ASW) - 15.59
FKRA = Flesch-Kincaid Reading Age
ASL = Average Sentence Length (i.e., the number of words divided by the number of sentences)
ASW = Average number of Syllable per Word (i.e., the number of syllables divided by the number of words)
Analyzing the results is a simple exercise. For instance, a score of 5.0 indicates a grade-school level; i.e., a score of 9.3 means that a ninth grader would be able to read the document. This score makes it easier for teachers, parents, librarians, and others to judge the readability level of various books and texts for the students.
Theoretically, the lowest grade level score could be -3.4, but since there are no real passages that have every sentence consisting of a one-syllable word, it is a highly improbable result in practice.

Flesch-Kincaid Grade Level Readability Formula is inbuilt within the MS-Word application. However, MS-Word doesn't score above grade 12 and any grade above 12 will be reported as Grade 12. The US Government Department of Defense uses Flesch-Kincaid Grade Level formula as a standard test.

c) The Coleman-Liau Readability Formula (also known as The Coleman-Liau Index)

The Coleman–Liau Readability Formula is a readability assessment test designed by linguists Meri Coleman and T. L. Liau to approximate the usability of a text. Coleman said he created the formula as one of the many ways to help the U.S. Office of Education calibrate the readability of all textbooks for the public school system. Like other popular readability formulas, the Coleman–Liau Index approximates a U.S. grade level to understand the text.

To grade a text **Coleman-Liau Readability Calculator** is available at:
http://www.readabilityformulas.com/free-readability-formula-tests.php

Similar to the **Automated Readability Index**, but unlike most of the other grade-level predictors, the **Coleman–Liau** relies on characters instead of syllables per word. Instead of using syllable/word and sentence length indices, Meri Coleman and T. L. Liau believed that computerized assessments understand characters more easily and accurately than counting syllables and sentence length.

Meri Coleman and T. L. Liau developed this formula to automatically (by computer) calculate samples of hard-copy text, instead of manually hard-coding the text. Unlike syllable-based readability indicators, it does not require you to analyze the characters that create the words (such as syllable counts) - only their length in characters. Therefore, you can scan a hardcopy of text into your word processor, use a free OCR program to recognize character, word, and sentence boundaries, and apply this formula to the text. According to Coleman, "There is no need to estimate syllables since word length in letters is a better predictor of readability than word length in syllables."

Calculate the Coleman–Liau Index with the following formula:
CLI = 0.0588L – 0.296S – 15.8
L is the average number of letters per 100 words.
S is the average number of sentences per 100 words.

As an example, let's use this abstract from a public domain book:

The best things in an artist's work are so much a matter of intuition, that there is much to be said for the point of view that would altogether discourage intellectual inquiry into artistic phenomena on the part of the artist. Intuitions are shy things and apt to disappear if looked into too closely. And there is undoubtedly a danger that too much knowledge and training may supplant the natural intuitive feeling of a student, leaving only a cold knowledge of the means of expression in its place. For the artist, if he has the right stuff in him ...

The abstract contains 4 sentences, 100 words, and 448 letters or digits; **L** = 448 and **S** = 4.

Formula: CLI = 0.0588 x 448(L) - 0.296 x 4.0(S) - 15.8 = 10.6
Therefore, this formula grades this abstract of text at a grade level of **10.6**, or roughly appropriate for a 10-11th grade high school student.

The Coleman–Liau Index grades this entire article (with the sample text) a **14.9**, which is appropriate for a college-level undergraduate student in the U.S.

Note: Although we used just 100 words for the sample above, you should input at least 300 words of text to calculate a more accurate grade score.
To grade a text **Coleman-Liau Readability Calculator** is available at:
http://www.readabilityformulas.com/free-readability-formula-tests.php

d) The Bormuth Readability Index - (Readability Formula):
Like other popular readability tests, the Bormuth Readability Index calculates a reading grade level required to read a text based on two factors:
1) Character count (average length of characters) rather than syllable count; and
2) Average number of familiar words in the sample text.
The Bormuth Readability Index uses the Dale-Chall word list to count familiar words in samples of text. Initially, Professor John R. Bormuth of the University of Chicago developed this formula to evaluate academic documents and school textbooks. This formula works reliably on texts above 4th grade level.

Typically, **the Bormuth Readability Index** outputs a grade level that closely matches the adjusted grade level from the new **Dale-Chall Readability Formula**. The differences between the two formulas are that the Bormuth Readability Index relies on character count instead of syllable count; and instead of calculating a percent of difficult words, it calculates the average of familiar words in the text.

The Bormuth Readability Index is sometimes called The Bormuth Formula, the Bormuth Grade Level Formula, or Degrees of Reading Power. The Bormuth Readability Index was adapted into the Degrees of Reading Power used by the College Entrance Examination Board in 1981.

Use the following formula to calculate the **Bormuth Readability Index**:
GL = 0.886593 - (**AWL** x 0.03640) + (**AFW** x 0.161911) - (**ASL** x 0.21401) – (**ASL** x 0.000577) - (**ASL** x 0.000005)
GL: Grade level needed to read the text.
AWL: Average word length or number of characters per word (number of characters divided by the number of words)

AFW: Average familiar words per word (the number of words in the Dale-Chall list of 3,000 simple words divided by the number of words)
ASL: Average sentence length in words or average number of words in sentence (number of words divided by the number of sentences)

The Bormuth Readability Index outputs a number that correlates to a U.S. grade level. For example, a result of 10.6 means students in 10th grade and above can read and comprehend the text. Unlike the new Dale-Chall Readability Formula which outputs an adjusted number that you must match to a number on an adjusted grade level table, the Bormuth Readability Index does not require you to use a table to determine an adjusted grade level.

7.4 SRS Important Ingredients:

An SRS document typically includes following important ingredients:

a) An SRS template
b) Identifying requirements and linking sources
c) A traceability matrix

a) An SRS Template:

The important step to writing an SRS is to select a template that can be used and fine tuned for the organizational needs. It is recommended to select an existing template or specification to begin with, and then adapt it to meet the project needs. A SRS outline and template is described at Annexure 8.

b) Identify and Link Requirements with Sources:

The SRS serves to define the functional and nonfunctional requirements of the product. Functional requirements each have an origin from which they came, be it a *use case* (which is used in system analysis to identify, clarify, and organize system requirements, and consists of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal), government regulation, industry standard, or a business requirement. In developing an SRS, identify these origins and link them to their corresponding requirements. Such a practice not only justifies the requirement, but it also helps assure project stakeholders that frivolous or spurious requirements are kept out of the specification.

To link requirements with their sources, each requirement included in the SRS should be labeled with a unique identifier that can remain valid over time as requirements are added, deleted, or changed. Such a labeling system helps maintain change-record integrity while also serving as an identification system for gathering metrics. You can begin a separate requirements identification list that ties a requirement identification (ID) number with a

description of the requirement. Eventually, that requirement ID and description become part of the SRS itself and then part of the Requirements Traceability Matrix.

c) Requirements Traceability Matrix (RTM):

The Requirements Traceability Matrix (RTM) functions as a sort of "chain of custody" document for requirements and can include pointers to links from requirements to sources, as well as pointers to business rules. For example, any given requirement must be traced back to a specified need, be it a use case, business essential, industry-recognized standard, or government regulation. As mentioned previously, linking requirements with sources minimizes or even eliminates the presence of spurious or frivolous requirements that lack any justification. The RTM is another record of mutual understanding, but also helps during the development phase.

As software design and development proceed, the design elements and the actual code must be tied back to the requirement(s) that define them. The RTM is filled as development progresses.

8. Annexure: Software Requirements Specification (SRS) Document Template

The following section gives a template for Software Requirements Specification (SRS) document. This template may be customized/ tailored as per the need and used for preparing SRS document.

Software Requirements Specification (SRS) Document Template

This document outline is based on the IEEE Standard 830-1998 for Software Requirements Specifications.
This document should specify what functions are to be performed on what data to produce what results at what location for whom.
A properly written SRS limits the range of valid designs, but does not specify any particular design.
A good SRS is

- Correct (accurately captures the “real” requirements)
- Unambiguous (all statements have exactly one interpretation)
- Complete (where TBDs are absolutely necessary, document why the information is unknown, who is responsible for resolution, and the deadline)
- Consistent
- Ranked for importance and/or stability
- Verifiable (avoid soft descriptions like “works well”, “is user friendly”; use concrete terms specify measurable quantities)
- Modifiable (evolve the SRS only via a formal change process, preserving a complete audit trail of changes)
- Traceable (cross reference with source documents and spawned documents)

The paragraphs written in the “Comment” style are for the benefit of the person writing the document and should be removed before the document is finalized.

CONTENTS

1. INTRODUCTION

[This section should provide an overview of the entire document.]

1.1 Purpose

[Describe the purpose of this specification and its intended audience.]

1.2 Scope

[This subsection should:
a) Identify the software product(s) to be produced by name (e.g., Host DBMS, Report Generator, etc.);
b) Explain what the software product(s) will, and, if necessary, will not do;
c) Describe the application of the software being specified, including relevant benefits, objectives, and goals;
d) Be consistent with similar statements in higher-level specifications (e.g., the system requirements specification), if they exist.]

1.3 Definitions, Acronyms, and Abbreviations

[This subsection should provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS.]

1.4 References

[This subsection should:
a) Provide a complete list of all documents referenced elsewhere in the SRS;
b) Identify each document by title, report number (if applicable), date, and publishing organization;
c) Specify the sources from which the references can be obtained.]

1.5 Overview

[This subsection should describe what the rest of the SRS contains, and explain how the SRS is organized.]

2. OVERALL DESCRIPTION

[In this section, describe the general factors that affect the product and its requirements. This section consists of six subsections described below

2.1 Product Perspective

This section should place the product in perspective with other related products. If the product is independent and self-contained, state that here. Otherwise, identify interfaces between the

*product and related systems. A **context diagram** showing the major components of the larger system, interconnections, and external inter-faces can be helpful. This subsection should also describe how the software operates inside various constraints. For example, these constraints could include:*

2.1.1 System Interfaces

[List each system interface and identify the related functionality of the product.]

2.1.2 User Interfaces

[Specify the logical characteristics of each interface between the software product and its users (e.g., required screen formats, report layouts, menu structures, or function keys).
Specify all the aspects of optimizing the interface with the person who must use the system (e.g., required functionality to provide long or short error messages). This could be a list of do's and don'ts describing how the system will appear to the user.]

2.1.3 Hardware Interfaces

[Specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes configuration characteristics (e.g., number of ports, instruction sets), what devices are to be supported, and protocols.]

2.1.4 Software Interfaces

[Specify the use of other required software products (e.g., a DBMS or operating system), and interfaces with other application systems. For each required software product, provide the following:

- *Name*
- *Mnemonic*
- *Specification Number*
- *Version Number*
- *Source*

For each interface, discuss the purpose of the interfacing software, and define the interface in terms of message format and content. For well-documented interfaces, simply provide a reference to the documentation.]

2.1.5 Communications Interfaces

[Specify any interfaces to communications such as local area networks, etc.]

2.1.6 Memory Constraints

[Specify any applicable characteristics and limits on RAM, disk space, etc.]

2.1.7 Operations

[Specify any normal and special operations required by the user, including:

- *Periods of interactive operations and periods of unattended operations*
- *Data processing support functions*
- *Backup and recovery operations]*

2.1.8 Site Adaptation Requirements

[Define requirements for any data or initialization sequences that are specific to a given site, mission, or operational mode. Specify features that should be modified to adapt the software to a particular installation.]

2.2 Product Functions

[This subsection of the SRS should provide a summary of the major functions that the software will perform:

a) The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time;

b) *Textual or graphical methods can be used to show the different functions and their relationships.]*

2.3 User Characteristics

[Describe the general characteristics of the intended users, including

- *Educational level*
- *Experience*
- *Technical expertise]*

2.4 Constraints

[This subsection of the SRS should provide a general description of any other items that will limit the developer's options. These include:

- *Regulatory policies*
- *Hardware limitations*
- *Interfaces to other applications*
- *Parallel operation*
- *Audit functions*
- *Control functions*
- *Higher-order language requirements*
- *Signal handshake protocols*
- *Reliability requirements*

- *Criticality of the application*
- *Safety and security considerations]*

2.5 Assumptions and Dependencies

[List factors that affect the requirements. These factors are not design constraints, but areas where future changes might drive change in the requirements.]

This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS.

For example, an assumption may be that a specific operating system will be available on the hardware designated for the software product. If, in fact, the operating system is not available, the SRS would then have to change accordingly.

3. SPECIFIC REQUIREMENTS

[This section should describe all software requirements at a sufficient level of detail for designers to design a system satisfying the requirements and testers to verity that the system satisfies requirements.

Every stated requirement should be externally perceivable by users, operators or other external systems.

At a minimum, these requirements should describe every input into the software, every output from the software, and every function performed by the software in response to an input or in support of an output.

All requirements should be uniquely identifiable (e.g., by number).

The remainder of this sample document is organized according to A.5 Template of SRS Section 3 Organized by Feature shown in the Annex of Std 830-1998]

3.1 External Interface Requirements

[Provide a detailed description of all inputs into and outputs from the software. This section should complement the interface descriptions under section 2.1 and should not repeat information there. Include both content and format as follows:

- *Name of item*
- *Description of purpose*
- *Source of input or destination of output*
- *Valid range, accuracy, and/or tolerance*
- *Units of measure*
- *Timing*
- *Relationships to other inputs/outputs*
- *Screen formats/organization*

- *Window formats/organization*
- *Data formats*
- *Command formats*
- *End messages*

These requirements may be organized in the following subsections.]

3.1.1 User Interfaces

3.1.2 Hardware Interfaces

3.1.3 Software Interfaces

3.1.4 Communications Interfaces

3.2 Software Product Features

[Features of the software should define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These include:

a) Validity checks on the inputs;

b) Exact sequence of operations;

c) Responses to abnormal situations, including

1) Overflow;

2) Communication facilities;

3) Error handling and recovery.

d) Effect of parameters;

e) Relationship of outputs to inputs, including

1) Input/output sequences;

2) Formulas for input to output conversion.]

3.3 Performance Requirements

[Specify static and dynamic numerical requirements placed on the software or on human interaction with the software.

Static numerical requirements may include the number of terminals to be supported, the number of simultaneous users to be supported, and the amount and type of information to be handled.

Dynamic numerical requirements may include the number of transactions and tasks and the amount of data to be processed within certain time period for both normal and peak workload conditions.

All of these requirements should be stated in measurable form.]

3.4 Logical Database Requirements

[Specify the requirements for any information that is to be placed into a database, including

- *Types of information used by various functions*
- *Frequency of use*
- *Accessing capabilities*
- *Data entities and their relationships*
- *Integrity constraints*
- *Data retention requirements]*

3.5 Design Constraints

[Specify requirements imposed by standards, hardware limitations, etc.]

3.6 Software System Attributes

[The following items provide a partial list of system attributes that can serve as requirements that should be objectively verified.]

3.6.1 Reliability

[Specify the factors needed to establish the software's required reliability.]

3.6.2 Availability

[Specify the factors needed to guarantee a defined level of availability.]

3.6.3 Security

[Specify the factors that will protect the software from accidental or malicious access, misuse, or modification. These factors may include:

- *Cryptography*
- *Activity logging*
- *Restrictions on inter module communications*
- *Data integrity checks]*

3.6.4 Maintainability

[Specify attributes of the software that relate to ease of maintenance. These requirements may relate to modularity, complexity, or interface design. Requirements should not be placed here simply because they are thought to be good design practices]

3.6.5 Portability

[Specify attributes of the software that relate to the ease of porting the software to other host machines and/or operating systems.]

3.7 Other Requirements