**IEEE Std 830-1998**

(Revision of

IEEE Std 830-1993)

**IEEE Recommended Practice for**

**Software Requirements Speci cations**

Sponsor

**Software Engineering Standards Committee of the**

**IEEE Computer Society**

Approved 25 June 1998

# IEEE-SA Standards Board

**Abstract:** The content and qualities of a good software requirements specification (SRS) are described and several sample SRS outlines are presented. This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products. Guidelines for compliance with IEEE/EIA 12207.1-1997 are also provided.

**Keywords:** contract, customer, prototyping, software requirements specification, supplier, system requirements specifications

The Institute of Electrical and Electronics Engineers, Inc. 345 East 47th Street, New York, NY 10017-2394, USA

Copyright ' 1998 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Published 1998. Printed in the United States of America.

ISBN 0-7381-0332-2

*No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.*

**IEEE Standards** documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every ve years for revision or reaf rmation. When a document is more than ve years old and has not been reaf rmed, it is reasonable to conclude that its contents, although still of some value, do not wholly re ect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership af liation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to speci c applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board

445 Hoes Lane

P.O. Box 1331

Piscataway, NJ 08855-1331

USA

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (978) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

# Introduction

(This introduction is not a part of IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Speci cations.)

This recommended practice describes recommended approaches for the speci cation of software requirements. It is based on a model in which the result of the software requirements speci cation process is an unambiguous and complete speci cation document. It should help

1. Software customers to accurately describe what they wish to obtain;
2. Software suppliers to understand exactly what the customer wants;
3. Individuals to accomplish the following goals:
   1. Develop a standard software requirements speci cation (SRS) outline for their own organizations;
   2. De ne the format and content of their speci c software requirements speci cations;
   3. Develop additional local supporting items such as an SRS quality checklist, or an SRS writer s handbook.

To the customers, suppliers, and other individuals, a good SRS should provide several speci c bene ts, such as the following:

*Establish the basis for agreement between the customers and the suppliers on what the software product is to do.* The complete description of the functions to be performed by the software speci ed in the SRS will assist the potential users to determine if the software speci ed meets their needs or how the software must be modi ed to meet their needs.

*Reduce the development effort.* The preparation of the SRS forces the various concerned groups in the customer s organization to consider rigorously all of the requirements before design begins and reduces later redesign, recoding, and retesting. Careful review of the requirements in the SRS can reveal omissions, misunderstandings, and inconsistencies early in the development cycle when these problems are easier to correct.

*Provide a basis for estimating costs and schedules.* The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates.

*Provide a baseline for validation and veri cation.* Organizations can develop their validation and veri cation plans much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured.

*Facilitate transfer.* The SRS makes it easier to transfer the software product to new users or new machines. Customers thus nd it easier to transfer the software to other parts of their organization, and suppliers nd it easier to transfer it to new customers.

*Serve as a basis for enhancement.* Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the nished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation.

The readers of this document are referred to Annex B for guidelines for using this recommended practice to meet the requirements of IEEE/EIA 12207.1-1997, IEEE/EIA Guide Industry Implementation of ISO/IEC 12207: 1995, Standard for Information Technology Software life cycle processes Life cycle data.

iii

## Participants

This recommended practice was prepared by the Life Cycle Data Harmonization Working Group of the Software Engineering Standards Committee of the IEEE Computer Society. At the time this recommended prac-

|  |  |  |
| --- | --- | --- |
| **Leonard L. Tripp,** *Chair* | |  |
| Edward Byrne Dennis Lawrence  Paul R. Croll David Maibor  Perry DeWeese Ray Milovanovic  Robin Fralick James Moore  Marilyn Ginsberg-Finner Timothy Niesen  John Harauz Dennis Rilling Mark Henley  The following persons were on the balloting committee: | | Terry Rout  Richard Schmidt  Norman F. Schneidewind  David Schultz  Basil Sherlund  Peter Voldner  Ronald Wade |
| Syed Ali  Theodore K. Atchinson  Mikhail Auguston  Robert E. Barry  Leo Beltracchi  H. Ronald Berlack  Richard E. Biehl  Michael A. Blackledge  Sandro Bologna  Juris Borzovs  Kathleen L. Briggs  M. Scott Buck  Michael Caldwell  James E. Cardow  Enrico A. Carrara  Lawrence Catchpole  Keith Chan  Antonio M. Cicu  Theo Clarke  Sylvain Clermont  Rosemary Coleman  Virgil Lee Cooper  W. W. Geoff Cozens  Paul R. Croll  Gregory T. Daich  Geoffrey Darnton  Taz Daughtrey  Bostjan K. Derganc  Perry R. DeWeese  James Do  Evelyn S. Dow  Carl Einar Dragstedt  Sherman Eagles  Christof Ebert  Leo Egan  Richard E. Fairley  John W. Fendrich  Jay Forster  Kirby Fortenberry  Eva Freund  Richard C. Fries  Roger U. Fujii  Adel N. Ghannam  Marilyn Ginsberg-Finner  John Garth Glynn  Julio Gonzalez-Sanz  L. M. Gunther | David A. Gustafson  Jon D. Hagar  John Harauz  Robert T. Harley  Herbert Hecht  William He ey  Manfred Hein  Mark Heinrich  Mark Henley  Debra Herrmann  John W. Horch  Jerry Huller  Peter L. Hung  George Jackelen  Frank V. Jorgensen  William S. Junk  George X. Kambic  Richard Karcich  Ron S. Kenett  Judith S. Kerner  Robert J. Kierzyk  Dwayne L. Knirk  Shaye Koenig  Thomas M. Kurihara  John B. Lane  J. Dennis Lawrence  Fang Ching Lim  William M. Lively  James J. Longbucco  Dieter Look  John Lord  Stan Magee  David Maibor  Harold Mains  Robert A. Martin  Tomoo Matsubara  Mike McAndrew  Patrick D. McCray  Christopher McMacken  Jerome W. Mersky  Bret Michael  Alan Miller  Celia H. Modell  James W. Moore  Pavol Navrat  Myrna L. Olson | Indradeb P. Pal  Alex Polack  Peter T. Poon  Lawrence S. Przybylski  Kenneth R. Ptack  Annette D. Reilly  Dennis Rilling  Andrew P. Sage  Helmut Sandmayr  Stephen R. Schach  Hans Schaefer  Norman Schneidewind  David J. Schultz  Lisa A. Selmon  Robert W. Shillato  David M. Siefert  Carl A. Singer  James M. Sivak  Richard S. Sky  Nancy M. Smith  Melford E. Smyre  Harry M. Sneed  Alfred R. Sorkowitz  Donald W. Sova  Luca Spotorno  Julia Stesney  Fred J. Strauss  Christine Brown Strysik  Toru Takeshita  Richard H. Thayer  Booker Thomas  Patricia Trellue  Theodore J. Urbanowicz  Glenn D. Venables  Udo Voges  David D. Walden  Dolores Wallace  William M. Walsh  John W. Walz  Camille SWhite-Partain  Scott A. Whitmire  P. A. Wolfgang  Paul R. Work  Natalie C. Yopconka  Janusz Zalewski  Geraldine Zimmerman  Peter F. Zoll |

tice was approved, the working group consisted of the following members:

iv

When the IEEE-SA Standards Board approved this recommended practice on 25 June 1998, it had the following membership:

|  |  |  |
| --- | --- | --- |
|  | **Judith Gorman,** *Secretary* | |
| Satish K. Aggarwal  Clyde R. Camp  James T. Carlo  Gary R. Engmann  Harold E. Epstein  Jay Forster\*  Thomas F. Garrity Ruben D. Garzon  \*Member Emeritus | James H. Gurney  Jim D. Isaak  Lowell G. Johnson  Robert Kennelly  E. G. Al Kiener  Joseph L. Koep nger\*  Stephen R. Lambert  Jim Logothetis  Donald C. Loughry  Valerie E. Zelenty  *IEEE Standards Project Editor* | L. Bruce McClung  Louis-Fran ois Pau  Ronald C. Petersen  Gerald H. Peterson  John B. Posey  Gary S. Robinson  Hans E. Weinrich  Donald W. Zipse |

**Richard J. Holleman,** *Chair* **Donald N. Heirman,** *Vice Chair*

v

# Contents

1. Overview.............................................................................................................................................. 1
   1. Scope............................................................................................................................................ 1
2. References............................................................................................................................................ 2
3. Definitions............................................................................................................................................ 2
4. Considerations for producing a good SRS........................................................................................... 3
   1. Nature of the SRS ........................................................................................................................ 3
   2. Environment of the SRS .............................................................................................................. 3
   3. Characteristics of a good SRS...................................................................................................... 4
   4. Joint preparation of the SRS........................................................................................................ 8
   5. SRS evolution .............................................................................................................................. 8
   6. Prototyping................................................................................................................................... 9
   7. Embedding design in the SRS...................................................................................................... 9
   8. Embedding project requirements in the SRS............................................................................. 10
5. The parts of an SRS ........................................................................................................................... 10
   1. Introduction (Section 1 of the SRS)........................................................................................... 11
   2. Overall description (Section 2 of the SRS)................................................................................ 12
   3. Specific requirements (Section 3 of the SRS)............................................................................ 15
   4. Supporting information.............................................................................................................. 19 Annex A (informative) SRS templates........................................................................................................ 21

Annex B (informative) Guidelines for compliance with IEEE/EIA 12207.1-1997.................................... 27

vi

**IEEE Recommended Practice for**

**Software Requirements Speci cations**

## 1. Overview

This recommended practice describes recommended approaches for the speci cation of software requirements. It is divided into ve clauses. Clause 1 explains the scope of this recommended practice. Clause 2 lists the references made to other standards. Clause 3 provides de nitions of speci c terms used. Clause 4 provides background information for writing a good SRS. Clause 5 discusses each of the essential parts of an SRS. This recommended practice also has two annexes, one which provides alternate format templates, and one which provides guidelines for compliance with IEEE/EIA 12207.1-1997.

### 1.1 Scope

This is a recommended practice for writing software requirements speci cations. It describes the content and qualities of a good software requirements speci cation (SRS) and presents several sample SRS outlines.

This recommended practice is aimed at specifying requirements of software to be developed but also can be applied to assist in the selection of in-house and commercial software products. However, application to already-developed software could be counterproductive.

When software is embedded in some larger system, such as medical equipment, then issues beyond those identi ed in this recommended practice may have to be addressed.

This recommended practice describes the process of creating a product and the content of the product. The product is an SRS. This recommended practice can be used to create such an SRS directly or can be used as a model for a more speci c standard.

This recommended practice does not identify any speci c method, nomenclature, or tool for preparing an SRS.

IEEE

## 2. References

This recommended practice shall be used in conjunction with the following publications.

ASTM E1340-96, Standard Guide for Rapid Prototyping of Computerized Systems.[[1]](#footnote-1)IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.[[2]](#footnote-2)

IEEE Std 730-1998, IEEE Standard for Software Quality Assurance Plans.

IEEE Std 730.1-1995, IEEE Guide for Software Quality Assurance Planning.

IEEE Std 828-1998, IEEE Standard for Software Con guration Management Plans.[[3]](#footnote-3)

IEEE Std 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software.

IEEE Std 982.2-1988, IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software.

IEEE Std 1002-1987 (Reaff 1992), IEEE Standard Taxonomy for Software Engineering Standards.

IEEE Std 1012-1998, IEEE Standard for Software Veri cation and Validation.

IEEE Std 1012a-1998, IEEE Standard for Software Veri cation and Validation: Content Map to IEEE/EIA

12207.1-1997.4

IEEE Std 1016-1998, IEEE Recommended Practice for Software Design Descriptions.5 IEEE Std 1028-1997, IEEE Standard for Software Reviews.

IEEE Std 1042-1987 (Reaff 1993), IEEE Guide to Software Con guration Management.

IEEE P1058/D2.1, Draft Standard for Software Project Management Plans, dated 5 August 1998.[[4]](#footnote-4)

IEEE Std 1058a-1998, IEEE Standard for Software Project Management Plans: Content Map to IEEE/EIA

12207.1-1997.[[5]](#footnote-5)

IEEE Std 1074-1997, IEEE Standard for Developing Software Life Cycle Processes.

IEEE Std 1233, 1998 Edition, IEEE Guide for Developing System Requirements Speci cations.8

## 3. De nitions

In general the de nitions of terms used in this recommended practice conform to the de nitions provided in IEEE Std 610.12-1990. The de nitions below are key terms as they are used in this recommended practice.

**3.1 contract:** A legally binding document agreed upon by the customer and supplier. This includes the technical and organizational requirements, cost, and schedule for a product. A contract may also contain informal but useful information such as the commitments or expectations of the parties involved.

**3.2 customer:** The person, or persons, who pay for the product and usually (but not necessarily) decide the requirements. In the context of this recommended practice the customer and the supplier may be members of the same organization.

**3.3 supplier:** The person, or persons, who produce a product for a customer. In the context of this recommended practice, the customer and the supplier may be members of the same organization.

**3.4 user:** The person, or persons, who operate or interact directly with the product. The user(s) and the customer(s) are often not the same person(s).

## 4. Considerations for producing a good SRS

This clause provides background information that should be considered when writing an SRS. This includes the following:

1. Nature of the SRS;
2. Environment of the SRS;
3. Characteristics of a good SRS;
4. Joint preparation of the SRS;
5. SRS evolution;
6. Prototyping;
7. Embedding design in the SRS;
8. Embedding project requirements in the SRS.

### 4.1 Nature of the SRS

The SRS is a speci cation for a particular software product, program, or set of programs that performs certain functions in a speci c environment. The SRS may be written by one or more representatives of the supplier, one or more representatives of the customer, or by both. Subclause 4.4 recommends both.

The basic issues that the SRS writer(s) shall address are the following:

1. *Functionality.* What is the software supposed to do?
2. *External interfaces.* How does the software interact with people, the system s hardware, other hardware, and other software?
3. *Performance.* What is the speed, availability, response time, recovery time of various software functions, etc.?
4. *Attributes.* What are the portability, correctness, maintainability, security, etc. considerations?
5. *Design constraints imposed on an implementation.* Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

The SRS writer(s) should avoid placing either design or project requirements in the SRS.

For recommended contents of an SRS see Clause 5.

### 4.2 Environment of the SRS

It is important to consider the part that the SRS plays in the total project plan, which is de ned in IEEE Std 610.12-1990. The software may contain essentially all the functionality of the project or it may be part of a larger system. In the latter case typically there will be an SRS that will state the interfaces between the system and its software portion, and will place external performance and functionality requirements upon the software portion. Of course the SRS should then agree with and expand upon these system requirements.

IEEE Std 1074-1997 describes the steps in the software life cycle and the applicable inputs for each step. Other standards, such as those listed in Clause 2, relate to other parts of the software life cycle and so may complement software requirements.

Since the SRS has a speci c role to play in the software development process, the SRS writer(s) should be careful not to go beyond the bounds of that role. This means the SRS

1. Should correctly de ne all of the software requirements. A software requirement may exist because of the nature of the task to be solved or because of a special characteristic of the project.
2. Should not describe any design or implementation details. These should be described in the design stage of the project.
3. Should not impose additional constraints on the software. These are properly speci ed in other documents such as a software quality assurance plan.

Therefore, a properly written SRS limits the range of valid designs, but does not specify any particular design.

### 4.3 Characteristics of a good SRS

An SRS should be

1. Correct;
2. Unambiguous;
3. Complete;
4. Consistent;
5. Ranked for importance and/or stability;
6. Veri able;
7. Modi able;
8. Traceable.

#### 4.3.1 Correct

An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet.

There is no tool or procedure that ensures correctness. The SRS should be compared with any applicable superior speci cation, such as a system requirements speci cation, 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 re ects the actual needs. Traceability makes this procedure easier and less prone to error (see 4.3.8).

#### 4.3.2 Unambiguous

An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. As a minimum, this requires that each characteristic of the nal 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 speci c.

An SRS is an important part of the requirements process of the software life cycle and is used in design, implementation, project monitoring, veri cation and validation, and in training as described in IEEE Std 1074-1997. 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 speci cation for the developer may be counterproductive in that they diminish understanding to the user and vice versa.

Subclauses 4.3.2.1 through 4.3.2.3 recommend how to avoid ambiguity.

##### 4.3.2.1 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.

##### 4.3.2.2 Requirements speci cation languages

One way to avoid the ambiguity inherent in natural language is to write the SRS in a particular requirements speci cation 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 nd them unintelligible. Moreover, these languages tend to be better at expressing certain types of requirements and addressing certain types of systems. Thus, they may in uence the requirements in subtle ways.

##### 4.3.2.3 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 ows. 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.

#### 4.3.3 Complete

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

1. All signi cant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specication should be acknowledged and treated.

1. De nition 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.
2. Full labels and references to all gures, tables, and diagrams in the SRS and de nition of all terms and units of measure.

##### 4.3.3.1 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

1. A description of the conditions causing the TBD (e.g., why an answer is not known) so that the situation can be resolved;
2. A description of what must be done to eliminate the TBD, who is responsible for its elimination, and by when it must be eliminated.

#### 4.3.4 Consistent

Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements speci cation, then it is not correct (see 4.3.1).

##### 4.3.4.1 Internal consistency

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

1. The speci ed characteristics of real-world objects may con ict. For example,
   1. The format of an output report may be described in one requirement as tabular but in another as textual.
   2. One requirement may state that all lights shall be green while another may state that all lights shall be blue.
2. There may be logical or temporal con ict between two speci ed actions. For example,
   1. One requirement may specify that the program will add two inputs and another may specify that the program will multiply them.
   2. One requirement may state that A must always follow B, while another may require that A and B occur simultaneously.
3. 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 de nitions promotes consistency.

#### 4.3.5 Ranked for importance and/or stability

An SRS is ranked for importance and/or stability if each requirement in it has an identi er 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 identi ed to make these differences clear and explicit. Identifying the requirements in the following manner helps:

1. Have customers give more careful consideration to each requirement, which often clari es any hidden assumptions they may have.
2. Have developers make correct design decisions and devote appropriate levels of effort to the different parts of the software product.

##### 4.3.5.1 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.

##### 4.3.5.2 Degree of necessity

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

1. *Essential.* Implies that the software will not be acceptable unless these requirements are provided in an agreed manner.
2. *Conditional.* Implies that these are requirements that would enhance the software product, but would not make it unacceptable if they are absent.
3. *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.

#### 4.3.6 Veri able

An SRS is veri able if, and only if, every requirement stated therein is veri able. A requirement is veri able if, and only if, there exists some nite 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 veri able.

Nonveri able requirements include statements such as works well, good human interface, and shall usually happen. These requirements cannot be veri ed because it is impossible to de ne the terms good, well, or usually. The statement that the program shall never enter an in nite loop is nonveri able because the testing of this quality is theoretically impossible.

An example of a veri able statement is

*Output of the program shall be produced within 20 s of event* · *60% of the time; and shall be produced within 30 s of event* · *100% of the time.*

This statement can be veri ed 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.

#### 4.3.7 Modi able

An SRS is modi able 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. Modi ability generally requires an SRS to

1. Have a coherent and easy-to-use organization with a table of contents, an index, and explicit crossreferencing;
2. Not be redundant (i.e., the same requirement should not appear in more than one place in the SRS);
3. 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 modi able.

#### 4.3.8 Traceable

An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation. The following two types of traceability are recommended:

1. *Backward traceability (i.e., to previous stages of development).* This depends upon each requirement explicitly referencing its source in earlier documents.
2. *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 modi ed, it is essential to be able to ascertain the complete set of requirements that may be affected by those modi cations.

### 4.4 Joint preparation of the SRS

The software development process should begin with supplier and customer agreement on what the completed software must do. This agreement, in the form of an SRS, should be jointly prepared. This is important because usually neither the customer nor the supplier is quali ed to write a good SRS alone.

1. Customers usually do not understand the software design and development process well enough to write a usable SRS.
2. Suppliers usually do not understand the customer s problem and eld of endeavor well enough to specify requirements for a satisfactory system.

Therefore, the customer and the supplier should work together to produce a well-written and completely understood SRS.

A special situation exists when a system and its software are both being de ned concurrently. Then the functionality, interfaces, performance, and other attributes and constraints of the software are not prede ned, but rather are jointly de ned and subject to negotiation and change. This makes it more dif cult, but no less important, to meet the characteristics stated in 4.3. In particular, an SRS that does not comply with the requirements of its parent system speci cation is incorrect.

This recommended practice does not speci cally discuss style, language usage, or techniques of good writing. It is quite important, however, that an SRS be well written. General technical writing books can be used for guidance.

### 4.5 SRS evolution

The SRS may need to evolve as the development of the software product progresses. It may be impossible to specify some details at the time the project is initiated (e.g., it may be impossible to de ne all of the screen formats for an interactive program during the requirements phase). Additional changes may ensue as de ciencies, shortcomings, and inaccuracies are discovered in the SRS.

Two major considerations in this process are the following:

1. Requirements should be speci ed as completely and thoroughly as is known at the time, even if evolutionary revisions can be foreseen as inevitable. The fact that they are incomplete should be noted.
2. A formal change process should be initiated to identify, control, track, and report projected changes.

Approved changes in requirements should be incorporated in the SRS in such a way as to

* 1. Provide an accurate and complete audit trail of changes;
  2. Permit the review of current and superseded portions of the SRS.

### 4.6 Prototyping

Prototyping is used frequently during the requirements portion of a project. Many tools exist that allow a prototype, exhibiting some characteristics of a system, to be created very quickly and easily. See also ASTM E1340-96.

Prototypes are useful for the following reasons:

1. The customer may be more likely to view the prototype and react to it than to read the SRS and react to it. Thus, the prototype provides quick feedback.
2. The prototype displays unanticipated aspects of the systems behavior. Thus, it produces not only answers but also new questions. This helps reach closure on the SRS.
3. An SRS based on a prototype tends to undergo less change during development, thus shortening development time.

A prototype should be used as a way to elicit software requirements. Some characteristics such as screen or report formats can be extracteddirectlyfrom the prototype. Other requirements can be inferred by running experiments with the prototype.

### 4.7 Embedding design in the SRS

A requirement speci es an externally visible function or attribute of a system. A design describes a particular subcomponent of a system and/or its interfaces with other subcomponents. The SRS writer(s) should clearly distinguish between identifying required design constraints and projecting a speci c design. Note that every requirement in the SRS limits design alternatives. This does not mean, though, that every requirement is design.

The SRS should specify what functions are to be performed on what data to produce what results at what location for whom. The SRS should focus on the services to be performed. The SRS should not normally specify design items such as the following:

1. Partitioning the software into modules;
2. Allocating functions to the modules;
3. Describing the ow of information or control between modules;
4. Choosing data structures.

#### 4.7.1 Necessary design requirements

In special cases some requirements may severely restrict the design. For example, security or safety requirements may re ect directly into design such as the need to

1. Keep certain functions in separate modules;
2. Permit only limited communication between some areas of the program;
3. Check data integrity for critical variables.

Examples of valid design constraints are physical requirements, performance requirements, software development standards, and software quality assurance standards.

Therefore, the requirements should be stated from a purely external viewpoint. When using models to illustrate the requirements, remember that the model only indicates the external behavior, and does not specify a design.

### 4.8 Embedding project requirements in the SRS

The SRS should address the software product, not the process of producing the software product.

Project requirements represent an understanding between the customer and the supplier about contractual matters pertaining to production of software and thus should not be included in the SRS. These normally include items such as

1. Cost;
2. Delivery schedules;
3. Reporting procedures;
4. Software development methods;
5. Quality assurance;
6. Validation and veri cation criteria;
7. Acceptance procedures.

Project requirements are speci ed in other documents, typically in a software development plan, a software quality assurance plan, or a statement of work.

## 5. The parts of an SRS

This clause discusses each of the essential parts of the SRS. These parts are arranged in Figure 1 in an outline that can serve as an example for writing an SRS.

While an SRS does not have to follow this outline or use the names given here for its parts, a good SRS should include all the information discussed here.

|  |
| --- |
| **Table of Contents**   1. Introduction    1. Purpose    2. Scope    3. Definitions, acronyms, and abbreviations    4. References    5. Overview 2. Overall description    1. Product perspective    2. Product functions    3. User characteristics    4. Constraints    5. Assumptions and dependencies 3. Specific requirements (See 5.3.1 through 5.3.8 for explanations of possible specific requirements. See also Annex A for several different ways of organizing this section of the SRS.)   Appendixes  Index |

**Figure 1 Prototype SRS outline**

### 5.1 Introduction (Section 1 of the SRS)

The introduction of the SRS should provide an overview of the entire SRS. It should contain the following subsections:

1. Purpose;
2. Scope;
3. De nitions, acronyms, and abbreviations;
4. References;
5. Overview.

#### 5.1.1 Purpose (1.1 of the SRS)

This subsection should

1. Delineate the purpose of the SRS;
2. Specify the intended audience for the SRS.

#### 5.1.2 Scope (1.2 of the SRS)

This subsection should

1. Identify the software product(s) to be produced by name (e.g., Host DBMS, Report Generator, etc.);
2. Explain what the software product(s) will, and, if necessary, will not do;
3. Describe the application of the software being speci ed, including relevant bene ts, objectives, and goals;
4. Be consistent with similar statements in higher-level speci cations (e.g., the system requirements speci cation), if they exist.

#### 5.1.3 De nitions, acronyms, and abbreviations (1.3 of the SRS)

This subsection should provide the de nitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.

#### 5.1.4 References (1.4 of the SRS)

This subsection should

1. Provide a complete list of all documents referenced elsewhere in the SRS;
2. Identify each document by title, report number (if applicable), date, and publishing organization;
3. Specify the sources from which the references can be obtained.

This information may be provided by reference to an appendix or to another document.

#### 5.1.5 Overview (1.5 of the SRS)

This subsection should

1. Describe what the rest of the SRS contains;
2. Explain how the SRS is organized.

### 5.2 Overall description (Section 2 of the SRS)

This section of the SRS should describe the general factors that affect the product and its requirements. This section does not state speci c requirements. Instead, it provides a background for those requirements, which are de ned in detail in Section 3 of the SRS, and makes them easier to understand.

This section usually consists of six subsections, as follows:

1. Product perspective;
2. Product functions;
3. User characteristics;
4. Constraints;
5. Assumptions and dependencies;
6. Apportioning of requirements.

#### 5.2.1 Product perspective (2.1 of the SRS)

This subsection of the SRS should put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS de nes a product that is a component of a larger system, as frequently occurs, then this subsection should relate the requirements of that larger system to functionality of the software and should identify interfaces between that system and the software.

A block diagram showing the major components of the larger system, interconnections, and external interfaces can be helpful.

IEEE

Std 830-1998

This subsection should also describe how the software operates inside various constraints. For example, these constraints could include

1. System interfaces;
2. User interfaces;
3. Hardware interfaces;
4. Software interfaces;
5. Communications interfaces;
6. Memory;
7. Operations;
8. Site adaptation requirements.

##### 5.2.1.1 System interfaces

This should list each system interface and identify the functionality of the software to accomplish the system requirement and the interface description to match the system.

##### 5.2.1.2 User interfaces

This should specify the following:

1. *The logical characteristics of each interface between the software product and its users.* This includes those con guration characteristics (e.g., required screen formats, page or window layouts, content of any reports or menus, or availability of programmable function keys) necessary to accomplish the software requirements.
2. *All the aspects of optimizing the interface with the person who must use the system.* This may simply comprise a list of do s and don ts on how the system will appear to the user. One example may be a requirement for the option of long or short error messages. Like all others, these requirements should be veri able, e.g., a clerk typist grade 4 can do function *X* in *Z* min after 1 h of training rather than a typist can do function *X.* (This may also be speci ed in the Software System Attributes under a section titled Ease of Use.)

##### 5.2.1.3 Hardware interfaces

This should specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes con guration characteristics (number of ports, instruction sets, etc.). It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. For example, terminal support may specify full-screen support as opposed to line-by-line support. **5.2.1.4 Software interfaces**

This should specify the use of other required software products (e.g., a data management system, an operating system, or a mathematical package), and interfaces with other application systems (e.g., the linkage between an accounts receivable system and a general ledger system). For each required software product, the following should be provided:

Name;

Mnemonic;

Speci cation number;

Version number; Source.

For each interface, the following should be provided:

Discussion of the purpose of the interfacing software as related to this software product.

De nition of the interface in terms of message content and format. It is not necessary to detail any well-documented interface, but a reference to the document de ning the interface is required.

##### 5.2.1.5 Communications interfaces

This should specify the various interfaces to communications such as local network protocols, etc. **5.2.1.6 Memory constraints**

This should specify any applicable characteristics and limits on primary and secondary memory.

##### 5.2.1.7 Operations

This should specify the normal and special operations required by the user such as

1. The various modes of operations in the user organization (e.g., user-initiated operations);
2. Periods of interactive operations and periods of unattended operations;
3. Data processing support functions;
4. Backup and recovery operations.

NOTE This is sometimes speci ed as part of the User Interfaces section.

##### 5.2.1.8 Site adaptation requirements

This should

1. De ne the requirements for any data or initialization sequences that are speci c to a given site, mission, or operational mode (e.g., grid values, safety limits, etc.);
2. Specify the site or mission-related features that should be modi ed to adapt the software to a particular installation.

#### 5.2.2 Product functions (2.2 of the SRS)

This subsection of the SRS should provide a summary of the major functions that the software will perform. For example, an SRS for an accounting program may use this part to address customer account maintenance, customer statement, and invoice preparation without mentioning the vast amount of detail that each of those functions requires.

Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level speci cation (if one exists) that allocates particular functions to the software product. Note that for the sake of clarity

1. 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 rst time.
2. Textual or graphical methods can be used to show the different functions and their relationships. Such a diagram is not intended to show a design of a product, but simply shows the logical relationships among variables.

#### 5.2.3 User characteristics (2.3 of the SRS)

This subsection of the SRS should describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. It should not be used to state speci c requirements, but rather should provide the reasons why certain speci c requirements are later speci ed in Section 3 of the SRS.

#### 5.2.4 Constraints (2.4 of the SRS)

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

1. Regulatory policies;
2. Hardware limitations (e.g., signal timing requirements);
3. Interfaces to other applications;
4. Parallel operation;
5. Audit functions;
6. Control functions;
7. Higher-order language requirements;
8. Signal handshake protocols (e.g., XON-XOFF, ACK-NACK);
9. Reliability requirements;
10. Criticality of the application;
11. Safety and security considerations.

#### 5.2.5 Assumptions and dependencies (2.5 of the SRS)

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 speci c 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.

#### 5.2.6 Apportioning of requirements (2.6 of the SRS)

This subsection of the SRS should identify requirements that may be delayed until future versions of the system.

### 5.3 Speci c requirements (Section 3 of the SRS)

This section of the SRS should contain all of the software requirements to a level of detail suf cient to enable designers to design a system to satisfy those requirements, and testers to test that the system satis es those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input or in support of an output. As this is often the largest and most important part of the SRS, the following principles apply:

1. Speci c requirements should be stated in conformance with all the characteristics described in 4.3.
2. Speci c requirements should be cross-referenced to earlier documents that relate.
3. All requirements should be uniquely identi able.
4. Careful attention should be given to organizing the requirements to maximize readability.

Before examining speci c ways of organizing the requirements it is helpful to understand the various items that comprise requirements as described in 5.3.1 through 5.3.7.

#### 5.3.1 External interfaces

This should be a detailed description of all inputs into and outputs from the software system. It should complement the interface descriptions in 5.2 and should not repeat information there.

It should include both content and format as follows:

1. Name of item;
2. Description of purpose;
3. Source of input or destination of output;
4. Valid range, accuracy, and/or tolerance;
5. Units of measure;
6. Timing;
7. Relationships to other inputs/outputs;
8. Screen formats/organization;
9. Window formats/organization;
10. Data formats;
11. Command formats;
12. End messages.

#### 5.3.2 Functions

Functional requirements should de ne the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as shall statements starting with The system shall

These include

1. Validity checks on the inputs
2. Exact sequence of operations
3. Responses to abnormal situations, including
   1. Over ow
   2. Communication facilities
   3. Error handling and recovery
4. Effect of parameters
5. Relationship of outputs to inputs, including
   1. Input/output sequences
   2. Formulas for input to output conversion

It may be appropriate to partition the functional requirements into subfunctions or subprocesses. This does not imply that the software design will also be partitioned that way.

#### 5.3.3 Performance requirements

This subsection should specify both the static and the dynamic numerical requirements placed on the software or on human interaction with the software as a whole. Static numerical requirements may include the following:

1. The number of terminals to be supported;
2. The number of simultaneous users to be supported;
3. Amount and type of information to be handled.

Static numerical requirements are sometimes identi ed under a separate section entitled Capacity.

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

All of these requirements should be stated in measurable terms.

For example,

*95% of the transactions shall be processed in less than 1 s.*

rather than,

*An operator shall not have to wait for the transaction to complete.*

NOTE Numerical limits applied to one speci c function are normally speci ed as part of the processing subparagraph description of that function.

#### 5.3.4 Logical database requirements

This should specify the logical requirements for any information that is to be placed into a database. This may include the following:

1. Types of information used by various functions;
2. Frequency of use;
3. Accessing capabilities;
4. Data entities and their relationships;
5. Integrity constraints;
6. Data retention requirements.

#### 5.3.5 Design constraints

This should specify design constraints that can be imposed by other standards, hardware limitations, etc.

##### 5.3.5.1 Standards compliance

This subsection should specify the requirements derived from existing standards or regulations. They may include the following:

1. Report format;
2. Data naming;
3. Accounting procedures;
4. Audit tracing.

For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or nancial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace le with before and after values.

#### 5.3.6 Software system attributes

There are a number of attributes of software that can serve as requirements. It is important that required attributes be speci ed so that their achievement can be objectively veri ed. Subclauses 5.3.6.1 through 5.3.6.5 provide a partial list of examples.

##### 5.3.6.1 Reliability

This should specify the factors required to establish the required reliability of the software system at time of delivery.

##### 5.3.6.2 Availability

This should specify the factors required to guarantee a de ned availability level for the entire system such as checkpoint, recovery, and restart.

##### 5.3.6.3 Security

This should specify the factors that protect the software from accidental or malicious access, use, modi cation, destruction, or disclosure. Speci c requirements in this area could include the need to

1. Utilize certain cryptographical techniques;
2. Keep speci c log or history data sets;
3. Assign certain functions to different modules;
4. Restrict communications between some areas of the program;
5. Check data integrity for critical variables.

##### 5.3.6.4 Maintainability

This should specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices.

##### 5.3.6.5 Portability

This should specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include the following:

1. Percentage of components with host-dependent code;
2. Percentage of code that is host dependent;
3. Use of a proven portable language;
4. Use of a particular compiler or language subset;
5. Use of a particular operating system.

#### 5.3.7 Organizing the speci c requirements

For anything but trivial systems the detailed requirements tend to be extensive. For this reason, it is recommended that careful consideration be given to organizing these in a manner optimal for understanding. There is no one optimal organization for all systems. Different classes of systems lend themselves to different organizations of requirements in Section 3 of the SRS. Some of these organizations are described in 5.3.7.1 through 5.3.7.7.

##### 5.3.7.1 System mode

Some systems behave quite differently depending on the mode of operation. For example, a control system may have different sets of functions depending on its mode: training, normal, or emergency. When organizing this section by mode, the outline in A.1 or A.2 should be used. The choice depends on whether interfaces and performance are dependent on mode.

##### 5.3.7.2 User class

Some systems provide different sets of functions to different classes of users. For example, an elevator control system presents different capabilities to passengers, maintenance workers, and re ghters. When organizing this section by user class, the outline in A.3 should be used.

##### 5.3.7.3 Objects

Objects are real-world entities that have a counterpart within the system. For example, in a patient monitoring system, objects include patients, sensors, nurses, rooms, physicians, medicines, etc. Associated with each object is a set of attributes (of that object) and functions (performed by that object). These functions are also called services, methods, or processes. When organizing this section by object, the outline in A.4 should be used. Note that sets of objects may share attributes and services. These are grouped together as classes.

##### 5.3.7.4 Feature

A feature is an externally desired service by the system that may require a sequence of inputs to effect the desired result. For example, in a telephone system, features include local call, call forwarding, and conference call. Each feature is generally described in a sequence of stimulus-response pairs. When organizing this section by feature, the outline in A.5 should be used.

##### 5.3.7.5 Stimulus

Some systems can be best organized by describing their functions in terms of stimuli. For example, the functions of an automatic aircraft landing system may be organized into sections for loss of power, wind shear, sudden change in roll, vertical velocity excessive, etc. When organizing this section by stimulus, the outline in A.6 should be used.

##### 5.3.7.6 Response

Some systems can be best organized by describing all the functions in support of the generation of a response. For example, the functions of a personnel system may be organized into sections corresponding to all functions associated with generating paychecks, all functions associated with generating a current list of employees, etc. The outline in A.6 (with all occurrences of stimulus replaced with response) should be used. **5.3.7.7 Functional hierarchy**

When none of the above organizational schemes prove helpful, the overall functionality can be organized into a hierarchy of functions organized by either common inputs, common outputs, or common internal data access. Data ow diagrams and data dictionaries can be used to show the relationships between and among the functions and data. When organizing this section by functional hierarchy, the outline in A.7 should be used.

#### 5.3.8 Additional comments

Whenever a new SRS is contemplated, more than one of the organizational techniques given in 5.3.7.7 may be appropriate. In such cases, organize the speci c requirements for multiple hierarchies tailored to the speci c needs of the system under speci cation. For example, see A.8 for an organization combining user class and feature. Any additional requirements may be put in a separate section at the end of the SRS.

There are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organization. For example, when organizing by mode, nite state machines or state charts may prove helpful; when organizing by object, object-oriented analysis may prove helpful; when organizing by feature, stimulus-response sequences may prove helpful; and when organizing by functional hierarchy, data ow diagrams and data dictionaries may prove helpful.

In any of the outlines given in A.1 through A.8, those sections called Functional Requirement *i* may be described in native language (e.g., English), in pseudocode, in a system de nition language, or in four subsections titled: Introduction, Inputs, Processing, and Outputs.

### 5.4 Supporting information

The supporting information makes the SRS easier to use. It includes the following:

1. Table of contents;
2. Index;
3. Appendixes.

**5.4.1 Table of contents and index**

The table of contents and index are quite important and should follow general compositional practices.

#### 5.4.2 Appendixes

The appendixes are not always considered part of the actual SRS and are not always necessary. They may include

1. Sample input/output formats, descriptions of cost analysis studies, or results of user surveys;
2. Supporting or background information that can help the readers of the SRS;
3. A description of the problems to be solved by the software;
4. Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements.

When appendixes are included, the SRS should explicitly state whether or not the appendixes are to be considered part of the requirements.

# Annex A (informative) SRS templates

## A.1 Template of SRS Section 3 organized by mode: Version 1

1. Speci c requirements
   1. External interface requirements
      1. User interfaces
      2. Hardware interfaces
      3. Software interfaces
      4. Communications interfaces
   2. Functional requirements
      1. Mode 1
         1. Functional requirement 1.1

.

.

.

3.2.1.*n*  Functional requirement 1.*n*

* + 1. Mode 2

.

.

.

3.2.*m* Mode *m*

* + - 1. Functional requirement *m.*1

.

.

.

3.2.*m.n* Functional requirement *m.n*

* 1. Performance requirements
  2. Design constraints
  3. Software system attributes
  4. Other requirements

## A.2 Template of SRS Section 3 organized by mode: Version 2

1. Speci c requirements
   1. Functional requirements
      1. Mode 1
         1. External interfaces
            1. User interfaces
            2. Hardware interfaces
            3. Software interfaces
            4. Communications interfaces
         2. Functional requirements 3.1.1.2.1 Functional requirement 1

.

.

.

3.1.1.2.*n* Functional requirement *n*

* + - 1. Performance
    1. Mode 2

.

.

.

3.1.*m* Mode *m*

* 1. Design constraints
  2. Software system attributes
  3. Other requirements

## A.3 Template of SRS Section 3 organized by user class

1. Speci c requirements
   1. External interface requirements
      1. User interfaces
      2. Hardware interfaces
      3. Software interfaces
      4. Communications interfaces
   2. Functional requirements
      1. User class 1
         1. Functional requirement 1.1

.

.

.

3.2.1.*n* Functional requirement 1.*n*

* + 1. User class 2

.

.

.

3.2.*m*  User class *m*

* + - 1. Functional requirement *m.*1

.

.

.

3.2.*m.n* Functional requirement *m.n*

* 1. Performance requirements
  2. Design constraints
  3. Software system attributes
  4. Other requirements

## A.4 Template of SRS Section 3 organized by object

1. Speci c requirements
   1. External interface requirements
      1. User interfaces
      2. Hardware interfaces
      3. Software interfaces
      4. Communications interfaces
   2. Classes/Objects
      1. Class/Object 1
         1. Attributes (direct or inherited)
            1. Attribute 1

.

.

.

3.2.1.1.*n* Attribute *n*

* + - 1. Functions (services, methods, direct or inherited)
         1. Functional requirement 1.1

.

.

.

3.2.1.2.*m* Functional requirement 1.*m*

* + - 1. Messages (communications received or sent)
    1. Class/Object 2

.

.

.

3.2.*p* Class/Object *p*

* 1. Performance requirements
  2. Design constraints
  3. Software system attributes
  4. Other requirements

## A.5 Template of SRS Section 3 organized by feature

1. Speci c requirements
   1. External interface requirements
      1. User interfaces
      2. Hardware interfaces
      3. Software interfaces
      4. Communications interfaces
   2. System features
      1. System Feature 1
         1. Introduction/Purpose of feature
         2. Stimulus/Response sequence
         3. Associated functional requirements 3.2.1.3.1 Functional requirement 1

.

.

.

3.2.1.3.*n* Functional requirement *n*

* + 1. System feature 2

.

.

.

3.2.*m* System feature *m*

.

.

.

* 1. Performance requirements
  2. Design constraints
  3. Software system attributes
  4. Other requirements

## A.6 Template of SRS Section 3 organized by stimulus

1. Speci c requirements
   1. External interface requirements
      1. User interfaces
      2. Hardware interfaces
      3. Software interfaces
      4. Communications interfaces
   2. Functional requirements
      1. Stimulus 1
         1. Functional requirement 1.1

.

.

.

3.2.1.*n* Functional requirement 1.*n*

* + 1. Stimulus 2

.

.

.

3.2.*m* Stimulus m

* + - 1. Functional requirement *m.*1

.

.

.

3.2.*m.n* Functional requirement *m.n*

* 1. Performance requirements
  2. Design constraints
  3. Software system attributes
  4. Other requirements

## A.7 Template of SRS Section 3 organized by functional hierarchy

1. Speci c requirements
   1. External interface requirements
      1. User interfaces
      2. Hardware interfaces
      3. Software interfaces
      4. Communications interfaces
   2. Functional requirements
      1. Information ows
         1. Data ow diagram 1
            1. Data entities
            2. Pertinent processes
            3. Topology
         2. Data ow diagram 2
            1. Data entities
            2. Pertinent processes
            3. Topology

.

.

.

3.2.1.*n* Data ow diagram *n*

* + - * 1. Data entities
        2. Pertinent processes
        3. Topology
    1. Process descriptions
       1. Process 1
          1. Input data entities
          2. Algorithm or formula of process
          3. Affected data entities
       2. Process 2
          1. Input data entities
          2. Algorithm or formula of process
          3. Affected data entities

.

.

.

3.2.2.*m* Process *m*

* + - * 1. Input data entities
        2. Algorithm or formula of process
        3. Affected data entities
    1. Data construct speci cations
       1. Construct 1
          1. Record type
          2. Constituent elds
       2. Construct 2
          1. Record type3.2.3.2.2 Constituent elds

.

.

.

3.2.3.*p* Construct *p*

* + - * 1. Record type
        2. Constituent elds
    1. Data dictionary
       1. Data element 1
          1. Name
          2. Representation
          3. Units/Format
          4. Precision/Accuracy
          5. Range
       2. Data element 2
          1. Name
          2. Representation
          3. Units/Format
          4. Precision/Accuracy
          5. Range

.

.

.

3.2.4.*q* Data element *q*

* + - * 1. Name
        2. Representation
        3. Units/Format
        4. Precision/Accuracy
        5. Range
  1. Performance requirements
  2. Design constraints
  3. Software system attributes
  4. Other requirements

## A.8 Template of SRS Section 3 showing multiple organizations

1. Speci c requirements
   1. External interface requirements
      1. User interfaces
      2. Hardware interfaces
      3. Software interfaces
      4. Communications interfaces
   2. Functional requirements
      1. User class 1
         1. Feature 1.1
            1. Introduction/Purpose of feature
            2. Stimulus/Response sequence
            3. Associated functional requirements
         2. Feature 1.2
            1. Introduction/Purpose of feature
            2. Stimulus/Response sequence3.2.1.2.3 Associated functional requirements

.

.

.

3.2.1.*m* Feature 1.*m*

* + - * 1. Introduction/Purpose of feature
        2. Stimulus/Response sequence
        3. Associated functional requirements
    1. User class 2

.

.

.

3.2.*n* User class *n*

.

.

.

* 1. Performance requirements
  2. Design constraints
  3. Software system attributes
  4. Other requirements

# Annex B (informative) Guidelines for compliance with IEEE/EIA 12207.1-1997

## B.1 Overview

The Software Engineering Standards Committee (SESC) of the IEEE Computer Society has endorsed the policy of adopting international standards. In 1995, the international standard, ISO/IEC 12207, Information technology Software life cycle processes, was completed. The standard establishes a common framework for software life cycle processes, with well-de ned terminology, that can be referenced by the software industry.

In 1995 the SESC evaluated ISO/IEC 12207 and decided that the standard should be adopted and serve as the basis for life cycle processes within the IEEE Software Engineering Collection. The IEEE adaptation of ISO/IEC 12207 is IEEE/EIA 12207.0-1996. It contains ISO/IEC 12207 and the following additions: improved compliance approach, life cycle process objectives, life cycle data objectives, and errata.

The implementation of ISO/IEC 12207 within the IEEE also includes the following:

IEEE/EIA 12207.1-1997, IEEE/EIA Guide for Information Technology Software life cycle processes Life cycle data;

IEEE/EIA 12207.2-1997, IEEE/EIA Guide for Information Technology Software life cycle processes Implementation considerations; and

Additions to 11 SESC standards (i.e., IEEE Stds 730, 828, 829, 830, 1012, 1016, 1058, 1062, 1219, 1233, 1362) to de ne the correlation between the data produced by existing SESC standards and the data produced by the application of IEEE/EIA 12207.1-1997.

NOTE Although IEEE/EIA 12207.1-1997 is a guide, it also contains provisions for application as a standard with speci c compliance requirements. This annex treats 12207.1-1997 as a standard.

### B.1.1 Scope and purpose

Both IEEE Std 830-1998 and IEEE/EIA 12207.1-1997 place requirements on a Software Requirements Description Document. The purpose of this annex is to explain the relationship between the two sets of requirements so that users producing documents intended to comply with both standards may do so.

## B.2 Correlation

This clause explains the relationship between IEEE Std 830-1998 and IEEE/EIA 12207.0-1996 and IEEE/ EIA 12207.1-1997 in the following areas: terminology, process, and life cycle data.

### B.2.1 Terminology correlation

Both this recommended practice and IEEE/EIA 12207.0-1996 have similar semantics for the key terms of software, requirements, speci cation, supplier, developer, and maintainer. This recommended practice uses the term customer where IEEE/EIA 12207.0-1996 uses acquirer, and this recommended practice uses user where IEEE/EIA 12207.0-1996 uses operator.

### B.2.2 Process correlation

IEEE/EIA 12207.0-1996 uses a process-oriented approach for describing the de nition of a set of requirements for software. This recommended practice uses a product-oriented approach, where the product is a Software Requirements Description (SRD). There are natural process steps, namely the steps to create each portion of the SRD. These may be correlated with the process requirements of IEEE/EIA 12207.0-1996. The difference is that this recommended practice is focused on the development of software requirements whereas IEEE/EIA 12207.0-1996 provides an overall life cycle view and mentions Software Requirements Analysis as part of its Development Process. This recommended practice provides a greater level of detail on what is involved in the preparation of an SRD.

### B.2.3 Life cycle data correlation

IEEE/EIA 12207.0-1996 takes the viewpoint that the software requirements are derived from the system requirements. Therefore, it uses the term, description rather that speci cation to describe the software requirements. In a system in which software is a component, each requiring its own speci cation, there would be a System Requirements Speci cation (SRS) and one or more SRDs. If the term Software Requirements Speci cation had been used, there would be a confusion between an SRS referring to the system or software requirements. In the case where there is a stand-alone software system, IEEE/EIA 12207.1-1997 states If the software is a stand-alone system, then this document should be a speci cation.

## B.3 Content mapping

This clause provides details bearing on a claim that an SRS complying with this recommended practice would also achieve document compliance with the SRD described in IEEE/EIA 12207.1-1997. The requirements for document compliance are summarized in a single row of Table 1 of IEEE/EIA 12207.11997. That row is reproduced in Table B.1 of this recommended practice.

**Table B.1 Summary of requirements for an SRD excerpted from Table 1 of IEEE/EIA 12207.1-1997**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Information item** | **IEEE/EIA**  **12207.0-1996 Clause** | **Kind** | **IEEE/EIA**  **12207.1-1997 Clause** | **References** |
| Software  Requirements  Description | 5.1.1.4, 5.3.4.1,  5.3.4.2 | Description  (See note for 6.22.1 of IEEE/EIA 12207.1-1997.) | 6.22 | IEEE Std 830-1998; EIA/IEEE  J-STD-016, F.2.3, F.2.4; MILSTD 961D. Also see ISO/IEC 5806, 5807, 6593, 8631, 8790, and 11411 for guidance on use of notations. |

The requirements for document compliance are discussed in the following subclauses:

B.3.1 discusses compliance with the information requirements noted in column 2 of Table B.1 as prescribed by 5.1.1.4, 5.3.4.1, and 5.3.4.2 of IEEE/EIA 12207.0-1996.

B.3.2 discusses compliance with the generic content guideline (the kind of document) noted in column 3 of Table B.1 as a description . The generic content guidelines for a description appear in 5.1 of IEEE/EIA 12207.1-1997.

B.3.3 discusses compliance with the speci c requirements for a Software Requirements Description noted in column 4 of Table B.1 as prescribed by 6.22 of IEEE/EIA 12207.1-1997.

B.3.4 discusses compliance with the life cycle data objectives of Annex H of IEEE/EIA 12207.01996 as described in 4.2 of IEEE/EIA 12207.1-1997.

### B.3.1 Compliance with information requirements of IEEE/EIA 12207.0-1996

The information requirements for an SRD are those prescribed by 5.1.1.4, 5.3.4.1, and 5.3.4.2 of IEEE/EIA 12207.0-1996. The requirements are substantively identical to those considered in B.3.3 of this recommended practice.

### B.3.2 Compliance with generic content guidelines of IEEE/EIA 12207.1-1997

According to IEEE/EIA 12207.1-1997, the generic content guideline for an SRD is generally a description, as prescribed by 5.1 of IEEE/EIA 12207.1-1997. A complying description shall achieve the purpose stated in 5.1.1 and include the information listed in 5.1.2 of IEEE/EIA 12207.1-1997.

The purpose of a description is:

IEEE/EIA 12207.1-1997, subclause 5.1.1: Purpose: Describe a planned or actual function, design, performance, or process.

An SRD complying with this recommended practice would achieve the stated purpose.

Any description or speci cation complying with IEEE/EIA 12207.1-1997 shall satisfy the generic content requirements provided in 5.1.2 of that standard. Table B.2 of this recommended practice lists the generic content items and, where appropriate, references the clause of this recommended practice that requires the same information.

**Table B.2 Coverage of generic description requirements by IEEE Std 830-1998**

|  |  |  |
| --- | --- | --- |
| **IEEE/EIA 12207.1-1997 generic content** | **Corresponding clauses of IEEE Std 830-1998** | **Additions to requirements of IEEE Std 830-1998** |
| a) Date of issue and status |  | Date of issue and status shall be provided. |
| b) Scope | 5.1.1 Scope |  |
| c) Issuing organization |  | Issuing organization shall be identi ed. |
| d) References | 5.1.4 References |  |
| e) Context | 5.1.2 Scope |  |
| f) Notation for description | 4.3 Characteristics of a good SRS |  |
| g) Body | 5. The parts of an SRS |  |
| h) Summary | 5.1.1. Overview |  |
| i) Glossary | 5.1.3 De nitions |  |
| j) Change history |  | Change history for the SRD shall be provided or referenced. |

### B.3.3 Compliance with speci c content requirements of IEEE/EIA 12207.1-1997

The speci c content requirements for an SRD in IEEE/EIA 12207.1-1997 are prescribed by 6.22 of IEEE/ EIA 12207.1-1997. A compliant SRD shall achieve the purpose stated in 6.22.1 of IEEE/EIA 12207.1-1997.

The purpose of the SRD is:

IEEE/EIA 12207.1-1997, subclause 6.22.1: Purpose: Specify the requirements for a software item and the methods to be used to ensure that each requirement has been met. Used as the basis for design and quali cation testing of a software item.

An SRS complying with this recommended practice and meeting the additional requirements of Table B.3 of this recommended practice would achieve the stated purpose.

An SRD compliant with IEEE/EIA 12207.1-1997 shall satisfy the speci c content requirements provided in 6.22.3 and 6.22.4 of that standard. Table B.3 of this recommended practice lists the speci c content items and, where appropriate, references the clause of this recommended practice that requires the same information.

An SRD speci ed according the requirements stated or referenced in Table B.3 of this recommended practice shall be evaluated considering the criteria provided in 5.3.4.2 of IEEE/EIA 12207.0-1996.

**Table B.3 Coverage of speci c SRD requirements by IEEE Std 830-1998**

|  |  |  |
| --- | --- | --- |
| **IEEE/EIA 12207.1-1997 speci c content** | **Corresponding clauses of IEEE Std 830-1998** | **Additions to requirements of IEEE Std 830-1998** |
| a) Generic description information | See Table B.2 |  |
| b) System identi cation and overview | 5.1.1 Scope |  |
| c) Functionality of the software item including:  — Performance requirements  — Physical characteristics  — Environmental conditions | 5.3.2 Functions  5.3.3 Performance requirements | Physical characteristics and environmental conditions should be provided. |
| d) Requirements for interfaces external to software item | 5.3.1 External interfaces |  |
| e) Quali cation requirements |  | The requirements to be used for quali cation testing should be provided (or referenced). |
| f) Safety speci cations | 5.2.4 Constraints |  |
| g) Security and privacy speci cations | 5.3.6.3 Security |  |
| h) Human-factors engineering requirements | 5.2.3 User characteristics  5.2.1.2 User interfaces |  |
| i) Data de nition and database requirements | 5.3.4 Logical data base requirements |  |
| j) Installation and acceptance requirements at operation site | 5.2.1.8 Site adaptation requirements | Installation and acceptance requirements at operation site |
| k) Installation and acceptance requirements at maintenance site | Installation and acceptance require- | ments at maintenance site |
| l) User documentation requirements |  | User documentation requirements |
| m) User operation and execution requirements | 5.2.1.7 Operations | User execution requirements |

**Table B.3 Coverage of speci c SRD requirements by IEEE Std 830-1998 *(continued)***

|  |  |  |
| --- | --- | --- |
| **IEEE/EIA 12207.1-1997 speci c content** | **Corresponding clauses of IEEE Std 830-1998** | **Additions to requirements of IEEE Std 830-1998** |
| n) User maintenance requirements | 5.3.6.4 Maintainability |  |
| o) Software quality characteristics | 5.3.6 Software system attributes |  |
| p) Design and implementation constraints | 5.2.4 Constraints |  |
| q) Computer resource requirements | 5.3.3 Performance requirements | Computer resource requirements |
| r) Packaging requirements |  | Packaging requirements |
| s) Precedence and criticality of requirements | 5.2.6 Apportioning of requirements |  |
| t) Requirements traceability | 4.3.8 Traceable |  |
| u) Rationale | 5.2.5 Assumptions and dependencies |  |
| Items a) through f) below are from  6.22.4  a) Support the life cycle data objectives of Annex H of IEEE/EIA  12207.0-1996 |  | Support the life cycle data objectives of Annex H of IEEE/EIA 12207.01996 |
| b) Describe any function using wellde ned notation | 4.3 Characteristics of a good SRS |  |
| c) De ne no requirements that are in con ict | 4.3 Characteristics of a good SRS |  |
| d) User standard terminology and de nitions | 5.1.3 De nition |  |
| e) De ne each unique requirement one to prevent inconsistency | 4.3 Characteristics of a good SRS |  |
| f) Uniquely identify each requirement | 4.3 Characteristics of a good SRS |  |

### B.3.4 Compliance with life cycle data objectives

In addition to the content requirements, life cycle data shall be managed in accordance with the objectives provided in Annex H of IEEE/EIA 12207.0-1996.

## B.4 Conclusion

The analysis suggests that any SRS complying with this recommended practice and the additions shown in Table B.2 and Table B.3 also complies with the requirements of an SRD in IEEE/EIA 12207.1-1997. In addition, to comply with IEEE/EIA 12207.1-1997, an SRS shall support the life cycle data objectives of Annex H of IEEE/EIA 12207.0-1996.

1. ASTM publications are available from the American Society for Testing and Materials, 100 Barr Harbor Drive, West Conshohocken, PA 19428-2959, USA. [↑](#footnote-ref-1)
2. IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. [↑](#footnote-ref-2)
3. As this standard goes to press, IEEE Std 828-1998; IEEE Std 1012a-1998; IEEE Std 1016-1998; and IEEE Std 1233, 1998 Edition are approved but not yet published. The draft standards are, however, available from the IEEE. Anticipated publication date is Fall 1998. Contact the IEEE Standards Department at 1 (732) 562-3800 for status information. 4See Footnote 3. 5See Footnote 3. [↑](#footnote-ref-3)
4. Upon approval of IEEE P1058 by the IEEE-SA Standards Board, this standard will be integrated with IEEE Std 1058a-1998 and published as IEEE Std 1058, 1998 Edition. Approval is expected 8 December 1998. [↑](#footnote-ref-4)
5. As this standard goes to press, IEEE Std 1058a-1998 is approved but not yet published. The draft standard is, however, available from the IEEE. Anticipated publication date is December 1998. Contact the IEEE Standards Department at 1 (732) 562-3800 for status information. See Footnote 6. 8See Footnote 3. [↑](#footnote-ref-5)