IEEE Std 830-1993
(Revision of IEEE Std 830-1984)

IEEE Recommended Practice for Software Requirements Specifications

Sponsor

Software Engineering Standards Committee
of the
IEEE Computer Society

Approved December 2, 1993

IEEE 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.
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

ISBN 1-55937-395-4

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE 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 five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect 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 affiliation 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 specific 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 technical 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 Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

Introduction

(This introduction is not a part of IEEE Std 830-1993, IEEE Recommended Practice for Software Requirements Specifications.)

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

a) Software customers to accurately describe what they wish to obtain
b) Software suppliers to understand exactly what the customer wants
c) Individuals to accomplish the following goals:
    1) Develop a standard software requirements specification (SRS) outline for their own organizations
    2) Define the format and content of their specific software requirements specifications
    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 specific benefits, such as:

a) *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 specified in the SRS will assist the potential user to determine if the software specified meets their needs or how the software must be modified to meet their needs.
b) *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.
c) *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.
d) *Provide a baseline for validation and verification.* Organizations can develop their validation and verification 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.
e) *Facilitate transfer.* The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organization, and suppliers find it easier to transfer it to new customers.
f) *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 finished product. The SRS may need to be altered, but it does provide a foundation for continued production evaluation.

At the time this recommended practice was completed, the Software Requirements Specifications Working Group had the following membership:

Edward R. Byrne, *Chair*

Fletcher J. Buckley
A. M. Davis
J. W. Horch
Motti Klein
Thomas M. Kurihara
John R. Matras
Robert Nation
William C. Sasso
Carl S. Seddio
Carl A. Singer
Richard H. Thayer
Leonard L. Tripp
A. Yonda

The following persons were on the balloting committee:

D. Avery
H. R. Berlack
W. J. Boll, Jr.
F. Buckley
G. Cozens
S. Crawford
P. I. Davis
B. K. Derganc
C. L. Evans
J. W. Fendrich
K. Fortenberry
D. Gelperin
Y. Gershkovitch
J. Gonzalez
D. A. Gustafson
J. Harauz
W. Hefley
J. W. Horch
P. L. Hung
M. S. Karasik
R. Kosinski
T. Kurihara
R. Lane
B. Lau
B. Livson
J. Maayan
J. Marijarvi
R. Martin
S. D. Matthews
I. Mazza
S. McGrath
D. E. Nickle
G. D. Schumacher
R. W. Shillato
D. M. Siefert
H. M. Sneed
V. Srivastava
R. H. Thayer
G. D. Tice
L. L. Tripp
D. Wallace
W. M. Walsh
P. R. Work
N. C. Yopconka
J. Zalewski

When the IEEE Standards Board approved this standard on December 2, 1993, it had the following membership:

Wallace S. Read, *Chair*

Donald C. Loughry, *Vice Chair*

Andrew G. Salem, *Secretary*

Gilles A. Baril
José A. Berrios de la Paz
Clyde R. Camp
Donald C. Fleckenstein
Jay Forster*
David F. Franklin
Ramiro Garcia
Donald N. Heirman
Jim Isaak
Ben C. Johnson
Walter J. Karplus
Lorraine C. Kevra
E. G. "Al" Kiener
Ivor N. Knight
Joseph L. Koepfinger*
D. N. "Jim" Logothetis
Don T. Michael*
Marco W. Migliaro
L. John Rankine
Arthur K. Reilly
Ronald H. Reimer
Gary S. Robinson
Leonard L. Tripp
Donald W. Zipse

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall
Richard B. Engelman
David E. Soffrin
Stanley I. Warshaw

Rachel A. Meisel
IEEE Standards Project Editor

Contents

THIS PAGE WAS
BLANK IN THE ORIGINAL

IEEE Recommended Practice for Software Requirements Specifications

1. Overview

This recommended practice describes recommended approaches for the specification of software requirements. It is divided into five clauses. Clause 1 explains the scope of this recommended practice. Clause 2 lists the references made to other standards. Clause 3 provides definitions of specific 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 an annex, which provides alternate format templates.

1.1 Scope

This is a recommended practice for writing software requirements specifications. It describes the content and qualities of a good software requirements specification (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 identified in this standard may have to be addressed.

This recommended practice describes the process of creating a product and the content of the product. The product is a software requirements specification. This recommended practice can be used to create such software requirements specification directly or can be used as a model for a more specific standard.

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

2. References

ASTM 1340-90, Standard Guide for Rapid Prototyping of Computerized Systems.[1]

IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology (ANSI).[2]

IEEE Std 730-1989, IEEE Standard for Software Quality Assurance Plans (ANSI).

IEEE Std 828-1990, IEEE Standard for Software Configuration Management Plans (ANSI).

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

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

IEEE Std 983-1986, IEEE Guide to Software Quality Assurance Planning.[3]

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

IEEE Std 1012-1986, IEEE Standard for Software Verification and Validation Plans (ANSI).

IEEE Std 1016-1987, IEEE Recommended Practice for Software Design Descriptions (ANSI).

IEEE Std 1028-1988, IEEE Standard for Software Reviews and Audits (ANSI).

IEEE Std 1042-1987, IEEE Guide to Software Configuration Management (ANSI).

IEEE Std 1058.1-1987, IEEE Standard for Project Management Plans (ANSI).

IEEE Std 1074-1991, IEEE Standard for Developing Software Life Cycle Processes (ANSI).

IEEE P1233, October 1993, Draft Guide to Developing System Requirements Specifications.[4]

[1]ASTM publications are available from the Customer Service Department, American Society for Testing and Materials, 1916 Race Street, Philadelphia, PA 19103, USA.

[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.

[3]This standard has been withdrawn; however, copies can be obtained from the IEEE Standards Department, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

[4]This authorized standards project was not approved by the IEEE Standards Board at the time this went to press. It is available from the IEEE Standards Department, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

3. Definitions

In general the definitions of terms used in this recommended practice conform to the definitions provided in IEEE Std 610.12-1990.[5] The definitions 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 document, 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).

[5]Information on references can be found in clause 2.

4. Considerations for producing a good SRS

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

a) Nature of the SRS
b) Environment of the SRS
c) Characteristics of a good SRS
d) Joint preparation of the SRS
e) SRS evolution
f) Prototyping
g) Embedding design in the SRS
h) Embedding project requirements in the SRS

4.1 Nature of the SRS

The SRS is a specification for a particular software product, program, or set of programs that performs certain functions in a specific 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:

a) *Functionality.* What is the software supposed to do?
b) *External interfaces.* How does the software interact with people, the system's hardware, other hardware, and other software?
c) *Performance.* What is the speed, availability, response time, recovery time of various software functions, etc.?
d) *Attributes.* What are the portability, correctness, maintainability, security, etc. considerations?
e) *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 defined 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 1074-1991 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 specific role to play in the software development process, SRS writer(s) should be careful not to go beyond the bounds of that role. This means the SRS

a) Should correctly define 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.

b) Should not describe any design or implementation details. These should be described in the design stage of the project.
c) Should not impose additional constraints on the software. These are properly specified 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

a) Correct
b) Unambiguous
c) Complete
d) Consistent
e) Ranked for importance and/or stability
f) Verifiable
g) Modifiable
h) 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 assures 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 assure 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 (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 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 as described in IEEE Std 1074-1991. 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.

The following subclauses recommend how to avoid ambiguity.

4.3.2.1 Natural language pitfalls

Requirements are often written in natural language (for example, 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 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 non-technical 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.

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 dataflows. 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:

a) *All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces.* In particular any external requirements placed 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.*

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

a) A description of the conditions causing the TBD (for example, 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

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 specification, 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 conflict. There are three types of likely conflicts in an SRS:

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 textual.
   2) One requirement may state that all lights shall be green while another states that all lights shall be blue.

c) There may be logical or temporal conflict between two specified 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 requires that "A and B" occur simultaneously.

d) 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.

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 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.

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.

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.

4.3.6 Verifiable

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 s of event x 60% of the time; and shall be produced within 30 s 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.

4.3.7 Modifiable

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; that is, 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.

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. Two types of traceability are recommended.

a) *Backward traceability (that is, to previous stages of development).* This depends upon each requirement explicitly referencing its source in earlier documents.
b) *Forward traceability (that is, 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.

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 qualified to write a good SRS alone.

a) Customers usually do not understand the software design and development process well enough to write a usable SRS.
b) Suppliers usually do not understand the customer's problem and field 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 defined concurrently. Then the functionality, interfaces, performance, and other attributes and constraints of the software are not predefined but rather are jointly defined and subject to negotiation and change. This makes it more difficult, but no less important, to meet the characteristics stated in 4.3. In particular, a SRS that does not comply with the requirements of its parent system specification is incorrect.

This recommended practice does not specifically 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. For example, it may be impossible to define all of the screen formats for an interactive program during the requirements phase. Additional changes may ensue as deficiencies, shortcomings, and inaccuracies are discovered in the SRS.

Two major considerations in this process are the following:

a) Requirements should be specified 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.
b) 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 being 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 Std 1340-90.

Prototypes are useful for three reasons:

a) 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.
b) 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.
c) 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 taken right from the prototype. Other requirements can be inferred by running experiments with the prototype.

4.7 Embedding design in the SRS

A requirement specifies 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 specific 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:

a) Partitioning the software into modules
b) Allocating functions to the modules
c) Describing the flow of information or control between modules
d) 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 reflect directly into design such as the need to

a) Keep certain functions in separate modules
b) Permit only limited communication between some areas of the program
c) 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 pure external viewpoint. When using models to illustrate the requirements, remember that the model just 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 customer and supplier about contractual matters pertaining to production of software and thus should not be included in the SRS. These normally include such items as

a) Cost
b) Delivery schedules
c) Reporting procedures
d) Software development methods
e) Quality assurance
f) Validation and verification criteria
g) Acceptance procedures

Project requirements are specified 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.

Table of Contents

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Overall description
   - 2.1 Product perspective
   - 2.2 Product functions
   - 2.3 User characteristics
   - 2.4 Constraints
   - 2.5 Assumptions and dependencies
3. Specific requirements (See 5.3.1–5.3.8 for explanations of possible specific requirements. See also annex A for several different organizations of this section of the SRS.)

Appendixes

Index

Figure 1—Prototype SRS Outline

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.

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:

a) Purpose
b) Scope
c) Definitions, acronyms, and abbreviations
d) References
e) Overview

5.1.1 Purpose (1.1 of the SRS)

This subsection should accomplish the following:

a) Delineate the purpose of the particular SRS
b) Specify the intended audience for the SRS

5.1.2 Scope (1.2 of the SRS)

This subsection should

a) Identify the software product(s) to be produced by name (for example, 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 (for example, the system requirement specification), if they exist

5.1.3 Definitions, acronyms, and abbreviations (1.3 of the SRS)

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

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

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

a) Describe what the rest of the SRS contains
b) 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 specific requirements. Instead, it provides a background for those requirements, which are defined in detail in section 3, and makes them easier to understand.

This section usually consists of six subsections, as follows:

a) Product perspective
b) Product functions
c) User characteristics
d) Constraints
e) Assumptions and dependencies
f) Requirements subsets

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 defines 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.

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

a) System interfaces
b) User interfaces
c) Hardware interfaces
d) Software interfaces
e) Communications interfaces
f) Memory constraints
g) Operations
h) 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:

a) *The logical characteristics of each interface between the software product and its users.* This includes those configuration 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.
b) *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 verifiable, for example, "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 specified 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 configuration 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.

5.2.1.4 Software interfaces

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

a) Name
b) Mnemonic
c) Specification number
d) Version number
e) Source

For each interface, the following should be provided:

a) Discussion of the purpose of the interfacing software as related to this software product.
b) Definition 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 defining 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

a) The various modes of operations in the user organization; for example user-initiated operations
b) Periods of interactive operations and periods of unattended operations
c) Data processing support functions
d) Backup and recovery operations

NOTE—This is sometimes specified as part of the User Interfaces section.

5.2.1.8 Site adaptation requirements

This could

a) Define the requirements for any data or initialization sequences that are specific to a given site, mission, or operational mode, for example, grid values, safety limits, etc.
b) Specify the site or mission-related features that should be modified 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 specification (if one exists) that allocates particular functions to the software product. Note that for the sake of clarity:

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. 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 specific requirements but rather should provide the reasons why certain specific requirements are later specified 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

a) Regulatory policies
b) Hardware limitations (for example, signal timing requirements)
c) Interfaces to other applications
d) Parallel operation
e) Audit functions
f) Control functions
g) Higher-order language requirements
h) Signal handshake protocols (for example, XON-XOFF, ACK-NACK)
i) Reliability requirements
j) Criticality of the application
k) 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 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.

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 Specific requirements (Section 3 of the SRS)

This section of the SRS should contain all the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies 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:

a) Specific requirements should be stated in conformance with all the characteristics described in 4.3 of this recommended practice.
b) Specific requirements should be cross-referenced to earlier documents that relate.
c) All requirements should be uniquely identifiable.
d) Careful attention should be given to organizing the requirements to maximize readability.

Before examining specific ways of organizing the requirements it is helpful to understand the various items that comprise requirements as described in the following subclauses.

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:

a) Name of item
b) Description of purpose
c) Source of input or destination of output
d) Valid range, accuracy and/or tolerance
e) Units of measure
f) Timing
g) Relationships to other inputs/outputs
h) Screen formats/organization
i) Window formats/organization
j) Data formats
k) Command formats
l) End messages

5.3.2 Functions

Functional requirements 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 are generally listed as "shall" statements starting with *The system shall...*

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

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:

a) The number of terminals to be supported
b) The number of simultaneous users to be supported
c) Amount and type of information to be handled

Static numerical requirements are sometimes identified 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 specific function are normally specified 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:

a) Types of information used by various functions
b) Frequency of use
c) Accessing capabilities
d) Data entities and their relationships
e) Integrity constraints
f) 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

a) Report format
b) Data naming
c) Accounting procedures
d) 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 financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file 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 specified so that their achievement can be objectively verified. The following items 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 defined availability level for the entire system such as checkpoint, recovery, and restart.

5.3.6.3 Security

In order to reach the goals highlighted in the \hyperref[sec:goals]{goals section} the system need to interface with Databases and DBMSs, required in order to store data about users, violations, and the corresponding metadata

This should specify the factors that would protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to

a) Utilize certain cryptographical techniques
b) Keep specific log or history data sets
c) Assign certain functions to different modules
d) Restrict communications between some areas of the program
e) 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

a) Percentage of components with host-dependent code
b) Percentage of code that is host dependent
c) Use of a proven portable language
d) Use of a particular compiler or language subset
e) Use of a particular operating system

5.3.7 Organizing the specific 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 the following subclauses.

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, use the outline shown in A.1 or A.2 of annex A. 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 fire fighters. When organizing this section by user class, use the outline shown in A.3 in annex A.

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, use the outline shown in A.4 in annex A. 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, use the outline shown in A.5 in annex A.

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, use the outline shown in A.6 in annex A.

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. Use A.6 in annex A (with all occurrences of *stimulus* replaced with *response*).

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 flow 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, use the outline in A.7 in annex A.

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 specific requirements for multiple hierarchies tailored to the specific needs of the system under specification. For example, see A.8 in annex A 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, finite 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; when organizing by functional hierarchy, data flow diagrams and data dictionaries may prove helpful.

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

5.4 Supporting information

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

a) Table of contents
b) Index
c) 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 requirements specification and are not always necessary. They may include

a) Sample I/O formats, descriptions of cost analysis studies, or results of user surveys
b) Supporting or background information that can help the readers of the SRS
c) A description of the problems to be solved by the software
d) 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)

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

3 Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 Mode 1
    - 3.2.1.1 Functional requirement 1.1
    - .
    - .
    - .
    - 3.2.1.*n* Functional requirement 1.*n*
  - 3.2.2 Mode 2
  - .
  - .
  - .
  - 3.2.*m* Mode *m*
    - 3.2.*m*.1 Functional requirement *m*.1
    - .
    - .
    - .
    - 3.2.*m*.*n* Functional requirement *m*.*n*
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

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

3. Specific requirements
- 3.1. Functional requirements
  - 3.1.1 Mode 1
    - 3.1.1.1 External interfaces
      - 3.1.1.1.1 User interfaces
      - 3.1.1.1.2 Hardware interfaces
      - 3.1.1.1.3 Software interfaces
      - 3.1.1.1.4 Communications interfaces
    - 3.1.1.2 Functional requirements
      - 3.1.1.2.1 Functional requirement 1
      - .
      - .
      - .
      - 3.1.1.2.*n* Functional requirement *n*
    - 3.1.1.3 Performance

3.1.2 Mode 2
.
.
.
3.1.*m* Mode *m*
3.2 Design constraints
3.3 Software system attributes
3.4 Other requirements

A.3 Template of SRS section 3 organized by user class

3 Specific requirements
3.1 External interface requirements
3.1.1 User interfaces
3.1.2 Hardware interfaces
3.1.3 Software interfaces
3.1.4 Communications interfaces
3.2 Functional requirements
3.2.1 User class 1
3.2.1.1 Functional requirement 1.1
.
.
.
3.2.1.*n* Functional requirement 1.*n*
3.2.2 User class 2
.
.
.
3.2.*m* User class *m*
3.2.*m*.1 Functional requirement *m*.1
.
.
.
3.2.*m.n* Functional requirement *m.n*
3.3 Performance requirements
3.4 Design constraints
3.5 Software system attributes
3.6 Other requirements

A.4 Template of SRS section 3 organized by object

3 Specific requirements
3.1 External interface requirements
3.1.1 User interfaces
3.1.2 Hardware interfaces
3.1.3 Software interfaces
3.1.4 Communications interfaces
3.2 Classes/Objects
3.2.1 Class/Object 1
3.2.1.1 Attributes (direct or inherited)
3.2.1.1.1 Attribute 1

.
.
.
3.2.1.1.*n* Attribute *n*
3.2.1.2 Functions (services, methods, direct or inherited)
3.2.1.2.1 Functional requirement 1.1
.
.
.
3.2.1.2.*m* Functional requirement 1.*m*
3.2.1.3 Messages (communications received or sent)
3.2.2 Class/Object 2
.
.
.
3.2.*p* Class/Object *p*
3.3 Performance requirements
3.4 Design constraints
3.5 Software system attributes
3.6 Other requirements

A.5 Template of SRS section 3 organized by feature

3 Specific requirements
3.1 External interface requirements
3.1.1 User interfaces
3.1.2 Hardware interfaces
3.1.3 Software interfaces
3.1.4 Communications interfaces
3.2 System features
3.2.1 System Feature 1
3.2.1.1 Introduction/Purpose of feature
3.2.1.2 Stimulus/Response sequence
3.2.1.3 Associated functional requirements
3.2.1.3.1 Functional requirement 1
.
.
.
3.2.1.3.*n* Functional requirement *n*
3.2.2 System feature 2
.
.
.
3.2.*m* System feature *m*
.
.
.
3.3 Performance requirements
3.4 Design constraints
3.5 Software system attributes
3.6 Other requirements

A.6 Template of SRS section 3 organized by stimulus

3 Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 Stimulus 1
    - 3.2.1.1 Functional requirement 1.1
    - .
    - .
    - .
    - 3.2.1.*n* Functional requirement 1.*n*
  - 3.2.2 Stimulus 2
  - .
  - .
  - .
  - 3.2.*m* Stimulus *m*
    - 3.2.*m*.1 Functional requirement *m*.1
    - .
    - .
    - .
    - 3.2.*m.n* Functional requirement *m.n*
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Software system attributes
- 3.6 Other requirements

A.7 Template of SRS section 3 organized by functional hierarchy

3 Specific requirements
- 3.1 External interface requirements
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communications interfaces
- 3.2 Functional requirements
  - 3.2.1 Information flows
    - 3.2.1.1 Data flow diagram 1
      - 3.2.1.1.1 Data entities
      - 3.2.1.1.2 Pertinent processes
      - 3.2.1.1.3 Topology
    - 3.2.1.2 Data flow diagram 2
      - 3.2.1.2.1 Data entities
      - 3.2.1.2.2 Pertinent processes
      - 3.2.1.2.3 Topology
    - .
    - .
    - .
    - 3.2.1.*n* Data flow diagram *n*

3.2.1.*n*.1 Data entities
3.2.1.*n*.2 Pertinent processes
3.2.1.*n*.3 Topology
3.2.2 Process descriptions
3.2.2.1 Process 1
3.2.2.1.1 Input data entities
3.2.2.1.2 Algorithm or formula of process
3.2.2.1.3 Affected data entities
3.2.2.2 Process 2
3.2.2.2.1 Input data entities
3.2.2.2.2 Algorithm or formula of process
3.2.2.2.3 Affected data entities
.
.
.
3.2.2.*m* Process *m*
3.2.2.*m*.1 Input data entities
3.2.2.*m*.2 Algorithm or formula of process
3.2.2.*m*.3 Affected data entities
3.2.3 Data construct specifications
3.2.3.1 Construct 1
3.2.3.1.1 Record type
3.2.3.1.2 Constituent fields
3.2.3.2 Construct 2
3.2.3.2.1 Record type
3.2.3.2.2 Constituent fields
.
.
.
3.2.3.*p* Construct *p*
3.2.3.*p*.1 Record type
3.2.3.*p*.2 Constituent fields
3.2.4 Data dictionary
3.2.4.1 Data element 1
3.2.4.1.1 Name
3.2.4.1.2 Representation
3.2.4.1.3 Units/Format
3.2.4.1.4 Precision/Accuracy
3.2.4.1.5 Range
3.2.4.2 Data element 2
3.2.4.2.1 Name
3.2.4.2.2 Representation
3.2.4.2.3 Units/Format
3.2.4.2.4 Precision/Accuracy
3.2.4.2.5 Range
.
.
.
3.2.4.*q* Data element *q*
3.2.4.*q*.1 Name
3.2.4.*q*.2 Representation
3.2.4.*q*.3 Units/Format
3.2.4.*q*.4 Precision/Accuracy
3.2.4.*q*.5 Range

3.3 Performance requirements
3.4 Design constraints
3.5 Software system attributes
3.6 Other requirements

A.8 Template of SRS section 3 showing multiple organizations

3 Specific requirements
3.1 External interface requirements
3.1.1 User interfaces
3.1.2 Hardware interfaces
3.1.3 Software interfaces
3.1.4 Communications interfaces
3.2 Functional requirements
3.2.1 User class 1
3.2.1.1 Feature 1.1
3.2.1.1.1 Introduction/Purpose of feature
3.2.1.1.2 Stimulus/Response sequence
3.2.1.1.3 Associated functional requirements
3.2.1.2 Feature 1.2
3.2.1.2.1 Introduction/Purpose of feature
3.2.1.2.2 Stimulus/Response sequence
3.2.1.2.3 Associated functional requirements
.
.
.
3.2.1.*m* Feature 1.*m*
3.2.1.*m*.1 Introduction/Purpose of feature
3.2.1.*m*.2 Stimulus/Response sequence
3.2.1.*m*.3 Associated functional requirements
3.2.2 User class 2
.
.
.
3.2.*n* User class *n*
.
.
.
3.3 Performance requirements
3.4 Design constraints
3.5 Software system attributes
3.6 Other requirements