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.

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 specification, such as a system requirements specification, with other project documentation, and

with other applicable standards, to ensure that it agrees. Alternatively the customer or user can determine if

the SRS correctly reflects the actual needs. Traceability makes this procedure easier and less prone to error

(see 4.3.8).

1. 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

1074-1997. 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 require-ments 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.

Subclauses 4.3.2.1 through 4.3.2.3 recommend how to avoid ambiguity.

1. Natural language pitfalls

Requirements are often written in natural language (e.g., English). Natural language is inherently ambigu-ous. A natural language SRS should be reviewed by an independent party to identify ambiguous use of

language so that it can be corrected.

1. 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.

1. Representation tools

In general, requirements methods and languages and the tools that support them fall into three general cate-gories—object, process, and behavioral. Object-oriented approaches organize the requirements in terms of

real-world objects, their attributes, and the services performed by those objects. Process-based approaches

organize the requirements into hierarchies of functions that communicate via data flows. Behavioral

approaches describe external behavior of the system in terms of some abstract notion (such as predicate

calculus), mathematical functions, or state machines.

The degree to which such tools and methods may be useful in preparing an SRS depends upon the size and

complexity of the program. No attempt is made here to describe or endorse any particular tool.

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

1. 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 imposed by a system speci-fication 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.

1. Use of TBDs

Any SRS that uses the phrase “to be determined” (TBD) is not a complete SRS. The TBD is, however, occa-sionally necessary and should be accompanied by

a) A description of the conditions causing the TBD (e.g., why an answer is not known) so that the situ-ation 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.

1. 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).

1. Internal consistency

An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict.

The three types of likely conflicts in an SRS are as follows:

a) The specified characteristics of real-world objects may conflict. For example,

1) The format of an output report may be described in one requirement as tabular but in another as

textual.

2) One requirement may state that all lights shall be green while another may state that all lights

shall be blue.

b) 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 may require that “A

and B” occur simultaneously.

c) Two or more requirements may describe the same real-world object but use different terms for that

object. For example, a program’s request for a user input may be called a “prompt” in one require-ment and a “cue” in another. The use of standard terminology and definitions promotes consistency.

1. 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 require-ments 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 differ-ent parts of the software product.

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.

1. 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.

1. 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 ¥ 60% of the time; and shall be

produced within 30 s of event ¥ 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.

1. 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 (i.e., the same requirement should not appear in more than one place in the SRS);

c) Express each requirement separately, rather than intermixed with other requirements.

Redundancy itself is not an error, but it can easily lead to errors. Redundancy can occasionally help to make

an SRS more readable, but a problem can arise when the redundant document is updated. For instance, a

requirement may be altered in only one of the places where it appears. The SRS then becomes inconsistent.

Whenever redundancy is necessary, the SRS should include explicit cross-references to make it modifiable.

1. 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 trace-ability are recommended:

a) Backward traceability (i.e., to previous stages of development).This depends upon each requirement

explicitly referencing its source in earlier documents.

b) Forward traceability (i.e., to all documents spawned by the SRS).This depends upon each require-ment 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.

TRANSLATE :

1 Benar   
Sebuah SRS benar jika, dan hanya jika, setiap kebutuhan yang dinyatakan di dalamnya adalah salah satu bahwa perangkat lunak harus memenuhi.   
Tidak ada alat atau prosedur yang menjamin kebenaran. SRS harus dibandingkan dengan yang berlaku   
spesifikasi unggul, seperti spesifikasi persyaratan sistem, dengan dokumentasi proyek lainnya, dan   
dengan standar yang berlaku lainnya, untuk memastikan bahwa itu setuju. Atau pelanggan atau pengguna dapat menentukan apakah   
SRS benar mencerminkan kebutuhan sebenarnya. Lacak membuat prosedur ini lebih mudah dan lebih rawan kesalahan   
(lihat 4.3.8).

2 Cukup jelas   
SRS adalah jelas jika, dan hanya jika, setiap kebutuhan yang dinyatakan di dalamnya hanya memiliki satu interpretasi. sebagai   
minimum, ini mensyaratkan bahwa setiap karakteristik dari produk akhir akan dijelaskan menggunakan istilah yang unik tunggal.   
Dalam kasus di mana istilah yang digunakan dalam konteks tertentu bisa memiliki beberapa arti, istilah harus dimasukkan   
dalam daftar istilah di mana artinya dibuat lebih spesifik.   
SRS adalah bagian penting dari proses persyaratan siklus hidup perangkat lunak dan digunakan dalam desain,   
pelaksanaan, monitoring proyek, verifikasi dan validasi, dan dalam pelatihan seperti yang dijelaskan dalam IEEE Std   
1074-1997. SRS harus jelas baik bagi mereka yang menciptakannya dan mereka yang menggunakannya. Namun,   
kelompok-kelompok ini sering tidak memiliki latar belakang yang sama dan karena itu tidak cenderung untuk menjelaskan perangkat lunak memerlukan-legal dengan cara yang sama. Representasi yang meningkatkan persyaratan spesifikasi untuk pengembang mungkin   
kontraproduktif dalam bahwa mereka mengurangi pemahaman kepada pengguna dan sebaliknya.   
Subclauses 4.3.2.1 melalui 4.3.2.3 merekomendasikan bagaimana untuk menghindari ambiguitas.   
a. Perangkap bahasa alami   
Persyaratan sering ditulis dalam bahasa alami (misalnya, bahasa Inggris). Bahasa alami secara inheren ambigu-ous. Sebuah bahasa alami SRS harus dikaji oleh pihak independen untuk mengidentifikasi penggunaan ambigu   
bahasa sehingga dapat diperbaiki.   
b. Persyaratan bahasa spesifikasi   
Salah satu cara untuk menghindari ambiguitas yang melekat dalam bahasa alami adalah menulis SRS dalam persyaratan tertentu   
spesifikasi bahasa. Prosesor bahasanya otomatis mendeteksi banyak leksikal, sintaksis, dan semantik   
kesalahan.   
Salah satu kelemahan dalam penggunaan bahasa tersebut adalah lamanya waktu yang dibutuhkan untuk mempelajarinya. Selain itu, banyak pengguna non-teknis menemukan mereka tidak dapat dimengerti. Selain itu, bahasa ini cenderung lebih baik dalam mengekspresikan tertentu   
jenis kebutuhan dan menangani beberapa jenis sistem. Dengan demikian, mereka dapat mempengaruhi persyaratan dalam   
cara yang halus.   
c. alat perwakilan   
Secara umum, metode persyaratan dan bahasa dan alat-alat yang mendukung mereka jatuh ke dalam tiga umum cate-gories-objek, proses, dan perilaku. Pendekatan berorientasi objek mengatur persyaratan dalam hal   
benda dunia nyata, atribut mereka, dan jasa yang dilakukan oleh objek tersebut. Proses berbasis pendekatan   
mengatur persyaratan dalam hirarki fungsi yang berkomunikasi melalui arus data. perilaku   
pendekatan menjelaskan perilaku eksternal dari sistem dalam hal beberapa gagasan abstrak (seperti predikat   
kalkulus), fungsi matematika, atau negara mesin.   
Sejauh mana alat dan metode tersebut mungkin berguna dalam mempersiapkan SRS tergantung pada ukuran dan   
kompleksitas program. Tidak ada usaha yang dibuat di sini untuk menggambarkan atau mendukung setiap alat tertentu.   
Ketika menggunakan pendekatan ini yang terbaik adalah mempertahankan deskripsi bahasa alami. Dengan begitu, custom-ers terbiasa dengan notasi masih bisa memahami SRS.

3 Complete   
Sebuah SRS selesai jika, dan hanya jika, itu termasuk unsur-unsur berikut:   
a) Semua persyaratan yang signifikan, baik yang berkaitan dengan kendala fungsi, kinerja, desain,   
atribut, atau antarmuka eksternal. Khususnya persyaratan eksternal yang dipaksakan oleh sistem spesifik-fikasi harus diakui dan diperlakukan.   
b) Definisi tanggapan perangkat lunak untuk semua kelas realisasi dari input data di semua realisasi   
kelas situasi. Perhatikan bahwa penting untuk menentukan tanggapan terhadap keduanya valid dan tidak valid masukan   
nilai-nilai.   
c) label lengkap dan referensi untuk semua gambar, tabel, dan diagram di SRS dan definisi dari semua istilah   
dan Satuan ukuran.   
a. Penggunaan TBDs   
Setiap SRS yang menggunakan frase "yang akan ditentukan" (TBD) bukanlah SRS lengkap. TBD adalah, bagaimanapun, occa-sionally diperlukan dan harus disertai dengan   
a) Penjelasan tentang kondisi yang menyebabkan TBD (misalnya, mengapa jawaban tidak diketahui) sehingga in situ-asi dapat diselesaikan;   
b) Penjelasan tentang apa yang harus dilakukan untuk menghilangkan TBD, yang bertanggung jawab untuk eliminasi, dan   
kapan itu harus dihilangkan.

4. Konsisten   
Konsistensi mengacu pada konsistensi internal. Jika SRS tidak setuju dengan beberapa dokumen-tingkat yang lebih tinggi, seperti   
sebagai spesifikasi persyaratan sistem, maka tidak benar (lihat 4.3.1).   
a. Konsistensi internal   
Sebuah SRS secara internal konsisten jika, dan hanya jika, tidak ada subset dari kebutuhan individu dijelaskan di dalamnya konflik.   
Ketiga jenis kemungkinan konflik dalam SRS adalah sebagai berikut:   
a) Karakteristik tertentu benda dunia nyata mungkin bertentangan. Sebagai contoh,   
1) Format laporan keluaran dapat digambarkan dalam satu persyaratan sebagaimana tabel tetapi di tempat lain sebagai   
tekstual.   
2) Salah satu persyaratan mungkin menyatakan bahwa semua lampu akan menjadi hijau sementara yang lain mungkin menyatakan bahwa semua lampu   
harus biru.   
b) Mungkin ada logis atau sementara konflik antara dua tindakan tertentu. Sebagai contoh,   
1) Salah satu persyaratan dapat menetapkan bahwa program ini akan menambah dua input dan satu lagi dapat menentukan   
bahwa program ini akan melipatgandakan mereka.   
2) Salah satu persyaratan mungkin menyatakan bahwa "A" harus selalu mengikuti "B," sementara yang lain mungkin mengharuskan "A   
dan B "terjadi secara bersamaan.   
c) Dua atau lebih persyaratan dapat menggambarkan objek dunia nyata yang sama tetapi menggunakan istilah yang berbeda untuk itu   
objek. Misalnya, permintaan program untuk input pengguna dapat disebut "prompt" dalam satu membutuhkan-ment dan "isyarat" di negara lain. Penggunaan terminologi standar dan definisi mempromosikan konsistensi.

5. Peringkat untuk kepentingan dan / atau stabilitas

SRS adalah peringkat untuk kepentingan dan / atau stabilitas jika setiap kebutuhan di dalamnya memiliki pengenal untuk menunjukkan baik

pentingnya atau stabilitas bahwa kebutuhan tertentu.

Biasanya, semua persyaratan yang berhubungan dengan produk perangkat lunak tidak sama pentingnya. Beberapa membutuhkan-KASIH mungkin penting, terutama untuk aplikasi kritis-hidup, sementara yang lain mungkin diinginkan.

Setiap persyaratan dalam SRS harus diidentifikasi untuk membuat perbedaan ini jelas dan eksplisit. mengidentifikasi

persyaratan dengan cara berikut membantu:

a) Memiliki pelanggan memberikan pertimbangan lebih berhati-hati untuk setiap kebutuhan, yang sering menjelaskan setiap

asumsi tersembunyi mereka mungkin memiliki.

b) Memiliki pengembang membuat keputusan desain yang benar dan mengabdikan tingkat yang tepat dari usaha untuk bagian berbeda-ent dari produk perangkat lunak.

a. Tingkat stabilitas

Salah satu metode persyaratan mengidentifikasi menggunakan dimensi stabilitas. Stabilitas dapat dinyatakan dalam istilah

dari jumlah perubahan yang diharapkan untuk kebutuhan apapun berdasarkan pengalaman atau pengetahuan yang akan datang

peristiwa yang mempengaruhi organisasi, fungsi, dan orang-orang yang didukung oleh sistem perangkat lunak.

b. Tingkat kebutuhan

Cara lain untuk peringkat persyaratan adalah untuk membedakan kelas persyaratan sebagai penting, kondisional, dan

opsional.

a) Essential.Implies bahwa perangkat lunak tidak akan diterima kecuali jika persyaratan ini diberikan dalam

cara yang disepakati.

b) Conditional.Implies bahwa ini adalah persyaratan yang akan meningkatkan produk perangkat lunak, tetapi akan

tidak membuatnya tidak dapat diterima jika mereka tidak hadir.

c) Optional.Implies kelas fungsi yang mungkin atau mungkin tidak berharga. Hal ini memberikan pemasok yang

kesempatan untuk mengusulkan sesuatu yang melebihi SRS....

6 diverifikasi

SRS dapat diverifikasi jika, dan hanya jika, setiap kebutuhan yang dinyatakan di dalamnya dapat diverifikasi. Sebuah persyaratan dapat diverifikasi

jika, dan hanya jika, terdapat beberapa proses hemat biaya hingga dengan mana seseorang atau mesin dapat memeriksa bahwa

produk perangkat lunak memenuhi persyaratan. Secara umum persyaratan ambigu tidak dapat diverifikasi.

Persyaratan Nonverifiable termasuk pernyataan seperti "bekerja dengan baik," "antarmuka manusia yang baik," dan "wajib

biasanya terjadi. "Persyaratan ini tidak dapat diverifikasi karena tidak mungkin untuk mendefinisikan istilah" baik, "

"Baik," atau "biasanya." Pernyataan bahwa "program tidak akan masuk loop tak terbatas" adalah nonverifiable

karena pengujian kualitas ini secara teoritis mungkin.

Contoh pernyataan diverifikasi adalah

Output program akan diproduksi dalam 20 s acara ¥ 60% dari waktu; dan harus

diproduksi dalam waktu 30 s acara ¥ 100% dari waktu.

Pernyataan ini dapat diverifikasi karena menggunakan konkret dan jumlah terukur.

Jika metode tidak dapat dirancang untuk menentukan apakah perangkat lunak memenuhi persyaratan tertentu, maka itu

persyaratan harus dihapus atau direvisi.

7 dimodifikasi

SRS adalah dimodifikasi jika, dan hanya jika, struktur dan gaya yang sedemikian rupa sehingga setiap perubahan persyaratan dapat

dibuat dengan mudah, benar-benar, dan konsisten sementara tetap mempertahankan struktur dan gaya. dapat dimodifikasi umumnya

membutuhkan SRS untuk

a) Memiliki organisasi yang koheren dan mudah digunakan dengan daftar isi, indeks, dan eksplisit referensi silang;

b) Tidak akan redundan (yaitu, persyaratan yang sama tidak akan muncul di lebih dari satu tempat di SRS);

c) Nyatakan setiap persyaratan secara terpisah, daripada bercampur dengan persyaratan lainnya.

Redundansi sendiri bukan kesalahan, tetapi dapat dengan mudah menyebabkan kesalahan. Redundansi kadang-kadang dapat membantu untuk membuat

SRS lebih mudah dibaca, tetapi masalah bisa timbul bila dokumen berlebihan diperbarui. Misalnya,

persyaratan dapat diubah hanya salah satu tempat di mana ia muncul. SRS kemudian menjadi tidak konsisten.

Setiap kali redundansi diperlukan, SRS harus mencakup eksplisit referensi silang untuk membuatnya dimodifikasi....

8 ditelusuri

SRS dapat dilacak jika asal masing-masing persyaratan jelas dan jika memfasilitasi referensi dari

kebutuhan masing-masing dalam pembangunan masa depan atau dokumentasi tambahan. Berikut dua jenis jejak-kemampuan yang direkomendasikan:

a) ketertelusuran Backward (yaitu, untuk tahap sebelumnya pembangunan) .Ini tergantung pada kebutuhan masing-masing

secara tegas menyebutkan sumbernya dalam dokumen sebelumnya.

b) Teruskan ketertelusuran (yaitu, untuk semua dokumen melahirkan oleh SRS) .Ini tergantung pada masing-masing memerlukan-ment di SRS memiliki nama yang unik atau nomor referensi.

Ketertelusuran maju dari SRS ini sangat penting ketika produk perangkat lunak memasuki operasi

dan fase pemeliharaan. Sebagai kode dan desain dokumen dimodifikasi, adalah penting untuk dapat memastikan

set lengkap persyaratan yang mungkin akan terpengaruh oleh mereka modifikasi.