3.3.1 Characteristics of an SRS

To properly satisfy the basic goals, an SRS should have certain properties and should contain different types of requirements. In this section, we discuss some of the desirable characteristics of an SRS and components of an SRS. A good SRS is [91, 92]:

1. Correct
2. Complete
3. Unambiguous
4. Verifiable
5. Consistent
6. Ranked for importance and/or stability
7. Modifiable
8. Traceable

The discussion of these properties here is based on [91, 92]. An SRS is *correct* if every requirement included in the SRS represents something required in the final system. An SRS is *complete* if everything the software is supposed to do and the responses of the software to all classes of input data are specified in the SRS. Correctness and completeness go hand-in-hand; whereas correctness ensures that which is specified is done correctly, completeness ensures that everything is indeed specified. Correctness is an easier property to establish than completeness as it basically involves examining each requirement to make sure it represents the user requirement. Completeness, on the other hand, is the most difficult property to establish; to ensure completeness, one has to detect the absence of specifications, and absence is much harder to ascertain than determining that what is present has some property.

An SRS is *unambiguous* if and only if every requirement stated has one and only one interpretation. Requirements are often written in natural language, which are inherently ambiguous. If the requirements are specified in a natural language, the SRS writer has to be especially careful to ensure that there are no ambiguities. One way to avoid ambiguities is to use some formal requirements specification language. The major disadvantage of using formal languages is the large effort required to write an SRS, the high cost of doing so, and the increased difficulty reading and understanding formally stated requirements (particularly by the users and clients).

An SRS is *verifiable* if and only if every stated requirement is verifiable. A requirement is verifiable if there exists some cost-effective process that can check whether the final software meets that requirement. This implies that the requirements should have as little subjectivity as possible because subjective requirements are difficult to verify. Unambiguity is essential for verifiability. As verification of requirements is often done through reviews, it also implies that an SRS is understandable, at least by the developer, the client, and the users. Understandability is clearly extremely important, as one of the goals of the requirements phase is to produce a document on which the client, the users, and the developers can agree.

An SRS is *consistent* if there is no requirement that conflicts with another. Terminology can cause inconsistencies; for example, different requirements may use different terms to refer to the same object. There may be logical or temporal conflict between requirements that causes inconsistencies. This occurs if the SRS contains two or more requirements whose logical or temporal characteristics cannot be satisfied together by any software system. For example, suppose a requirement states that an event e is to occur

before another event f. But then another set of requirements states (directly or indirectly by transitivity) that event f should occur before event e. Inconsistencies in an SRS can reflect of some major problems.

Generally, all the requirements for software are not of equal importance. Some are critical, others are important but not critical, and there are some which are desirable but not very important. Similarly, some requirements are "core" requirements which are not likely to change as time passes, while others are more dependent on time. An SRS is ranked for importance and/or stability if for each requirement the importance and the stability of the requirement are indicated. Stability of a requirement reflects the chances of it changing in future. It can be reflected in terms of the expected change volume.

Writing an SRS is an iterative process. Even when the requirements of a system are specified, they are later modified as the needs of the client change. Hence an SRS should be easy to modify. An SRS is *modifiable* if its structure and style are such that any necessary change can be made easily while preserving completeness and consistency. Presence of redundancy is a major hindrance to modifiability, as it can easily lead to errors. For example, assume that a requirement is stated in two places and that the requirement later needs to be changed. If only one occurrence of the requirement is modified, the resulting SRS will be inconsistent.

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 [91]. Forward traceability means that each requirement should be traceable to some design and code elements. Backward traceability requires that it be possible to trace design and code elements to the requirements they support. Traceability aids verification and validation.

Of all these characteristics, completeness is perhaps the most important (and hardest to ensure). One of the most common problem in requirements specification is when some of the requirements of the client are not specified. This necessitates additions and modifications to the requirements later in the development cycle, which are often expensive to incorporate. Incompleteness is also a major source of disagreement between the client and the supplier. The importance of having complete requirements cannot be overemphasized.

3.3.2 Components of an SRS

Completeness of specifications is difficult to achieve and even more difficult to verify. Having guidelines about what different things an SRS should specify

will help in completely specifying the requirements. Here we describe some of the system properties that an SRS should specify. The basic issues an SRS must address are:

- Functionality
- Performance
- Design constraints imposed on an implementation
- External interfaces

Conceptually, any SRS should have these components. If the traditional approach to requirement analysis is being followed, then the SRS might even have portions corresponding to these. However, functional requirements might be specified indirectly by specifying the services on the objects or by specifying the use cases.

Functional Requirements

Functional requirements specify which outputs should be produced from the given inputs. They describe the relationship between the input and output of the system. For each functional requirement, a detailed description of all the data inputs and their source, the units of measure, and the range of valid inputs must be specified.

All the operations to be performed on the input data to obtain the output should be specified. This includes specifying the validity checks on the input and output data, parameters affected by the operation, and equations or other logical operations that must be used to transform the inputs into corresponding outputs. For example, if there is a formula for computing the output, it should be specified. Care must be taken not to specify any algorithms that are not part of the system but that may be needed to implement the system. These decisions should be left for the designer.

An important part of the specification is the system behavior in abnormal situations, like invalid input (which can occur in many ways) or error during computation. The functional requirement must clearly state what the system should do if such situations occur. Specifically, it should specify the behavior of the system for invalid inputs and invalid outputs. Furthermore, behavior for situations where the input is valid but the normal operation cannot be performed should also be specified. An example of this situation is an airline reservation system, where a reservation cannot be made even for valid

passengers if the airplane is fully booked. In short, the system behavior for all foreseen inputs and all foreseen system states should be specified. These special conditions are often likely to be overlooked, resulting in a system that is not robust.

Performance Requirements

This part of an SRS specifies the performance constraints on the software system. All the requirements relating to the performance characteristics of the system must be clearly specified. There are two types of performance requirements: static and dynamic.

Static requirements are those that do not impose constraint on the execution characteristics of the system. These include requirements like the number of terminals to be supported, the number of simultaneous users to be supported, and the number of files that the system has to process and their sizes. These are also called *capacity* requirements of the system.

Dynamic requirements specify constraints on the execution behavior of the system. These typically include response time and throughput constraints on the system. Response time is the expected time for the completion of an operation under specified circumstances. Throughput is the expected number of operations that can be performed in a unit time. For example, the SRS may specify the number of transactions that must be processed per unit time, or what the response time for a particular command should be. Acceptable ranges of the different performance parameters should be specified, as well as acceptable performance for both normal and peak workload conditions.

All of these requirements should be stated in measurable terms. Requirements such as "response time should be good" or the system must be able to "process all the transactions quickly" are not desirable because they are imprecise and not verifiable. Instead, statements like "the response time of command x should be less than one second 90% of the times" or "a transaction should be processed in less than one second 98% of the times" should be used to declare performance specifications.

Design Constraints

There are a number of factors in the client's environment that may restrict the choices of a designer. Such factors include standards that must be followed, resource limits, operating environment, reliability and security re-

quirements, and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

Standards Compliance: This specifies the requirements for the standards the system must follow. The standards may include the report format and accounting procedures. There may be audit tracing requirements, which require certain kinds of changes, or operations that must be recorded in an audit file.

Hardware Limitations: The software may have to operate on some existing or predetermined hardware, thus imposing restrictions on the design. Hardware limitations can include the type of machines to be used, operating system available on the system, languages supported, and limits on primary and secondary storage.

Reliability and Fault Tolerance: Fault tolerance requirements can place a major constraint on how the system is to be designed. Fault tolerance requirements often make the system more complex and expensive. Requirements about system behavior in the face of certain kinds of faults is specified. Recovery requirements are often an integral part here, detailing what the system should do if some failure occurs to ensure certain properties. Reliability requirements are very important for critical applications.

Security: Security requirements are particularly significant in defense systems and many database systems. Security requirements place restrictions on the use of certain commands, control access to data, provide different kinds of access requirements for different people, require the use of passwords and cryptography techniques, and maintain a log of activities in the system. Given the current security needs even of common systems, they may also require proper assessment of security threats, proper programming techniques, and use of tools to detect flaws like buffer overflow.

External Interface Requirements

All the interactions of the software with people, hardware, and other software should be clearly specified. For the user interface, the characteristics of each user interface of the software product should be specified. User interface is becoming increasingly important and must be given proper attention. A preliminary user manual should be created with all user commands, screen formats, an explanation of how the system will appear to the user, and feedback and error messages. Like other specifications these requirements should be precise and verifiable. So, a statement like "the system should be user friendly" should be avoided and statements like "commands should

be no longer than six characters" or "command names should reflect the function they perform" used.

For hardware interface requirements, the SRS should specify the logical characteristics of each interface between the software product and the hardware components. If the software is to execute on existing hardware or on predetermined hardware, all the characteristics of the hardware, including memory restrictions, should be specified. In addition, the current use and load characteristics of the hardware should be given.

The interface requirement should specify the interface with other software the system will use or that will use the system. This includes the interface with the operating system and other applications. The message content and format of each interface should be specified.