Ricca, F., Scanniello, G., Torchiano, M., Reggio, G., and Astesiano, E. (2010). On the effectiveness of screen mockups in requirements engineering: Results from an internal replication. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 17:26.

Sutcliffe, A., Thew, S., and Jarvis, P. (2011). Experience with user-centred requirements engineering, *Requirements Engineering*, 16(4): 267–280.

Wilson, W.M., Rosenberg, L.H., and Hyatt, L.E. (1997). Automated analysis of requirement specifications. In *Proceedings of the 19th international Conference on Software Engineering*, 161–171.

Chapter 5 Requirements Validation

Requirements Risk Management

What Is Requirements Risk Management?

Poor requirements engineering practices have led to some spectacular failures. Bahill and Henderson (2005) analyzed several high-profile projects to determine if their failures were due to poor requirements development, requirements verification, or system validation. Their findings confirm that some of the most notorious system failures were due to poor requirements engineering. For example, the *Titanic* in 1912, the IBM PCjr in 1983, and the Mars Climate Orbiter in 1999 suffered from poor requirements design. The projects Apollo 13 in 1970, Space Shuttle *Challenger* in 1986, and Space Shuttle *Columbia* in 2002 all failed because of insufficient requirements verification (Table 5.1).

Requirements can be inadequate in many ways, including:

- Inaccurate or incomplete stakeholder identification
- Insufficient requirements validation
- Insufficient requirements verification
- Incomplete requirements
- Incorrect requirements
- Incorrectly ranked requirements
- Inconsistent requirements (Laplante 2010)

Requirements risk management involves the proactive identification, monitoring, and mitigation of any factors that can threaten the integrity of the requirements engineering process. Requirements risk factors can be divided into two

Table 5.1 A History of Project Failures Due to Poor Requirements Practices

System Name	Year	Requirements Process Failure
HMS Titanic	1912	Poor requirements design
Apollo 13	1970	Insufficient requirements verification
IBM PC jr	1983	Poor requirements design
Space Shuttle *Challenger*	1986	Insufficient requirements verification
Mars Climate Orbiter	1999	Poor requirements design
Space Shuttle *Columbia*	2002	Insufficient requirements verification

Source: Adapted from T.A. Bahill and S.J. Henderson, *Systems Engineering*, 8(1): 1–14, 2005.

types: technical and management. Technical risk factors pertaining to the elicitation, analysis, agreement, and representation processes have already been discussed in Chapters 3 and 4. In Chapter 6 we discuss the use of formal methods in improving the quality of requirements representation through mathematically rigorous approaches. Requirements management risk factors tend toward issues of expectation management and interpersonal relationships, and these are discussed in Chapter 9. In this chapter we focus on the mitigation of requirements risk through the analysis of the requirements specification document itself. That is, the validation and verification of the SRS should occur early in order to avoid costly problems further downstream. There is a variety of complementary and overlapping techniques to check quality attributes of the SRS, but IEEE Standard 830 contains a set of rules that are extremely helpful in vetting the technical aspects of the SRS and, in turn, mitigating risk later on. So in this chapter we look at the nature of SRS verification and validation, at various qualities of "goodness" for requirements specifications, and finally, turn to work at NASA that can be very helpful in comprehending these qualities of goodness.

To further motivate the notion of requirements risk, here is a vignette.*

On a road that the author travels, he passes a strange street sign that declares, "End brake retarder prohibition." The meaning of this sign is hard to understand, and is blurred by the curious use of a quadruple negative (each word in the directive has the connotation of stopping something). As it turns out, this sign is an exquisite example of a "shall not" requirement as well as illustrating a poor requirements

* A variation of this story first appeared in "End brake retarder prohibitions: Defining 'Shall Not' requirements effectively," *IT Professional*, May/June, 2010, pp. 46–52, by Jeff Voas and Phil Laplante.

specification, namely, one that is ambiguous, vague, contradictory, incomplete, or contains a mixed level of abstraction.

What is a "brake retarder"? Briefly, it is a device used on large trucks to slow the engine down by allowing air to be exhausted out of the pistons, thus slowing the vehicle. Brake retarders are very noisy, and, as a result, many municipalities prohibit their use within city limits. By Pennsylvania law, appropriate signs must be posted with the instructions, "Brake Retarders Prohibited within Municipal Limits." The complementary sign then reads "End Brake Retarder Prohibition." The latter phrasing is, apparently, unique to the Commonwealth of Pennsylvania (O'Neil 2004). In some states the sign pair reads "No Jake Brakes" and "End Jake Brake Prohibition," the term "Jake" being a nickname of a company that manufactures the device, Jacobs Vehicle Systems.

Let's have a little fun with the Pennsylvania form of the sign. If we substitute the synonyms:

end → stop, brake → stop, retarder → stopper, prohibition → stopping

the sign translates to:

"stop stop stopper stopping"

which is a quadruple negative. In theory, we should be able to replace such a quadruple negative with a null sign reading:

Ø

The original sign has the paradoxical effect, however, of causing a positive action (i.e., to allow the application of brake retarders).

Of course, this whimsical discussion ignores the fact that the word "end" is a verb, "retarder" and "prohibition" are nouns, and in this context "brake" is an adjective, rendering a logical analysis of meaning more difficult. And the sign really does have the intended effect: truck drivers know what the sign means. But it should be clear that this particular sign can be interpreted multiple ways and is, therefore, ambiguous.

The "brake retarder" sign is a poor one but perhaps not nearly as poor as the requirements for installing one. Pennsylvania's requirements on municipalities that wish to implement a "no brake retarder" zone are

1. Downhill grade(s) greater than 4%
2. A posted reduced speed limit for trucks due to a hazardous grade determination

3. Posted reduced gear zone(s)
4. Posted speed limits over 55 miles per hour
5. Highway exit ramps with a posted speed limit over 55 miles per hour

and the crash history for the stretch of road a municipality is seeking to keep brake-retarder free *must not include:*

1. *A history of runaway truck crashes over the past three years*
2. *A discernible pattern of rear-end crashes over the past three years where the truck was the striking vehicle* (O'Neil 2004)

The italic font was added to the latter section because it highlights the fact that the requirements on brake retarder prohibitions contain an embedded "shall not" requirement.

Given all of this confusion, clearly, a better job in formulating the language on the sign is needed, and more work is needed in formulating the requirements for the sign. Formal methods may have been advisable in this regard. All of these activities fall under the context of requirements risk management.

Requirements Validation and Verification

Requirements validation and verification involves review, analysis, and testing to ensure that a system complies with its requirements. Compliance pertains to both functional and nonfunctional requirements. But the foregoing definition does not readily distinguish between verification and validation and is probably too long in any case to be inspirational or to serve as a mission statement for the requirements engineer. Barry Boehm (1984) suggests the following to make the distinction between verification and validation:

- Requirements validation: "[A]m I building the right product?"
- Requirements verification: "[A]m I building the product right?"

In other words, validation involves fully understanding customer intent and verification involves satisfying customer intent.

There are terrific benefits to implementing a requirements verification and validation program. These include:

- Early detection and correction of system anomalies
- Enhanced management insight into process and product risk
- Support for life-cycle processes to ensure conformance to program performance and budget
- Early assessment of software and system performance

- Ability to obtain objective evidence of software and system conformance to support process
- Improved system development and maintenance processes
- Improved and integrated systems analysis model (IEEE 1012)

Requirements validation involves checking that the system provides all of the functions that best support the customer's needs and that it does not provide the functions that the customer does not need or want. Additionally, validation should ensure that there are no requirements conflicts and that satisfaction of the requirements can actually be demonstrated. Finally, there is some element of a sanity check in terms of time and budget: a requirement may be able to be literally met, but the cost and time needed to meet the requirement may be unacceptable or even impossible.

Requirements verification (testing) involves checking satisfaction of a number of desirable properties of the requirements (e.g., IEEE 830 rules). Usually we do both validation and verification (V&V) simultaneously, and often the techniques used for one or the other are the same.

Bahill and Henderson (2005) offer an informal verification/validation demonstration for a set of requirements for an electric water heater controller.

If 70° < Temperature < 100°, then the system shall output 3,000 watts.
If 100° < Temperature < 130°, then the system shall output 2,000 watts.
If 120° < Temperature < 150°, then the system shall output 1,000 watts.
If 150° < Temperature, then the system shall output 0 watts.

We notice that this set of requirements is incomplete because the behavior for Temperature < 70° is not defined. The requirements are also inconsistent: for example, what happens when Temperature = 125°? The requirements are also incorrect because the temperatures given are not specified as being in degrees Fahrenheit or Celsius. Clearly this set of requirements could have benefited from effective validation and verification processes.

Techniques for Requirements V&V

V&V techniques may include some of the requirements elicitation techniques that were discussed in Chapter 3. For example, group reviews/inspections, focus groups, prototyping, viewpoint resolution, or task analysis (through user stories and use cases) can be used to simplify, combine, or eliminate requirements. In addition, we can use comparative product evaluations to uncover missing or unreasonable requirements and task analysis to uncover and simplify requirements. Systematic manual analysis of the requirements, test-case generation (for testability and completeness), using an executable model of the system to check requirements, and automated consistency analysis may also be used. Finally, certain formal methods

such as model checking and consistency analysis can be used for V&V and these are discussed in Chapter 6. Let's look at some of these in greater detail.

Walkthroughs

Walkthroughs or peer reviews are an informal methodology that can be used to detect errors and improve requirements quality. The typical walkthrough scenario involves the requirements engineers, a supervisor, and peers participating in a semi-structured meeting to review the requirements prior to release. The goal is to identify ambiguities and inconsistencies and to determine if the requirements can be tested. The IEEE 830 rules can be used as a framework or checklist for the walkthrough.

Inspections

Inspections are a method of requirement quality control that can be informal (ad hoc) or highly structured, such as the one described shortly. More formal inspection processes, such as those described by Fagan (1986), provide close procedural control and repeatability. Fagan inspections define the following:

- What can be inspected
- When the code can be inspected
- Who can inspect the code
- What preparation is needed for the inspection
- How the inspection is to be conducted
- The data to be collected
- The follow-up activities

The Marinelli survey of 2008 gives some clues to the prevalence of the various inspection techniques in industry (Figure 5.1). For example, the survey showed both informal inspections and walkthroughs were used about 25% of the time each and about 20% reported using simple checklists or scenarios. Unfortunately no respondents reported using either automatic requirements inspection tools (we study one such tool later in this chapter) or formal inspections. In contrast, Sikora, Tenbergen, and Pohl (2012) interviewed 17 individuals from seven companies working in the area of embedded systems in Germany, and reported that all respondents were using or familiar with peer reviews, inspections, and formal methods.

Goal-Based Requirements Analysis

Stakeholders tend to express their requirements in terms of operations and actions rather than goals. A risk is posed when goals evolve as stakeholders change their

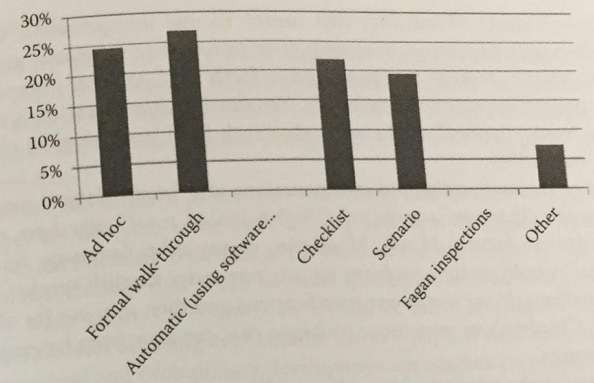

Figure 5.1 Requirements inspection technique used. (Modified from V. Marinelli, *An Analysis of Current Trends in Requirements Engineering Practice*, Master of Software Engineering Professional Paper, Penn State University, Great Valley School of Graduate Professional Studies, Malvern, PA, 2008.)

minds and refine and operationalize goals into behavioral requirements. To reduce this risk stakeholder goals need to be evolved until they can be structured as requirements.

Goal evolution is facilitated through goal elaboration and refinement. Useful techniques for goal elaborating include identifying goal obstacles, analyzing scenarios and constraints, and operationalizing goals. Goal refinement occurs when synonymous goals are reconciled, when goals are merged into a subgoal categorization, when constraints are identified, and when goals are operationalized (Hetzel 1988). As to operationalization, we refer to goals-based analysis when discussing metrics generation.

Requirements Understanding

In an early and influential book on software testing, Bill Hetzel (1988) proposed several paradigms for requirements verification and validation. To set the stage for this V&V, Hetzel addressed the problem of requirements understanding through the following analogy.

Imagine you are having a conversation with a "customer" in which he says that he would like for you to develop some kind of health management system, which, among other things, ensures that patients are eating a "well-balanced meal." You readily agree to this requirement. Many weeks later as you begin thinking about a system design, you reconsider the requirement to provide a

"well-balanced meal." What does that mean? In one interpretation, it could mean, adding up everything consumed: were minimum nutritional guidelines in terms of calories, protein, vitamins, and so forth met? Another interpretation is that the patient ate exactly three meals. Yet another interpretation is that the meals were "well balanced" in the sense that each food item weighed the same amount (Hetzel 1988).

Aside from the more ridiculous interpretations of a "well-balanced meal," the previous example illustrates a problem. "Well-balanced meal" may have no language equivalent in French, Hindi, Mandarin, or any other language. So "well-balanced meal" would create a problem for any nonnative English speaker. What other colloquialisms do we use in our specifications and then ship out for offshore development? Clearly, there are various problems that can arise from language and cultural differences.

One solution to the requirements understanding problem is offered by Hetzel. He suggests that for correct problem definition it is best to specify the test for accepting a solution along with the statement of the requirement. When the statement and test are listed together, most problems associated with misunderstanding requirements disappear. In particular we want to derive requirements-based test situations and use them as a test of requirement understanding and validation.

For example, when a requirement is found to be incomplete, we can use the test case to focus on missing information: that is, design the test case to ask the question, "What should the system do in this case when this input is not supplied?" Similarly, when a requirement is found to be fuzzy or imprecise, use a test case to ask the question, "Is this the result I should have in this situation?" The specific instance will focus attention on the imprecise answer or result and ensure that it is examined carefully (Hetzel 1988). Then an appropriate requirement can be introduced to deal with any lingering imprecision, ambiguity, or missing behavior. Today, Hetzel's approach would be called "test-driven development."

Validating Requirements Use Cases

When use cases comprise part of the requirements, these can be validated by asking a simple set of questions:

- Are there any additional actors that are not represented?
- Are there any activities that are not represented?
- Are each actor's goals being met?
- Are there events in the use case that do not address these goals?
- Can the use case be simplified?

Other related questions can and should be readily generated.

Prototyping

Prototypes are useful in V&V when very little is understood about the requirements or when it is necessary to gain some experience with the working model in order to discover requirements. The principle behind using working prototypes is the recognition that requirements change with experience and prototyping yields that experience.

There are two kinds of prototypes: throwaway and nonthrowaway. Throwaway prototypes are designed to be built quickly and then discarded. Nonthrowaway prototypes are intended to be used as the basis for the working code. The advantage of throwaway code is that it can be more quickly built. But nonthrowaway code does not waste effort. Of course, care should be taken to ensure that throwaway code does not end up being kept because it was likely not built robustly.

Incremental and evolutionary development approaches are essentially based on a series of nonthrowaway prototypes. The difference between the two approaches is, basically, that in incremental development the functionality of each release is planned, whereas in evolutionary development, subsequent releases are not planned out. In both incremental and evolutionary development, lessons learned from prior releases inform the functionality of future releases' incremental and evolutionary development. In essence, early versions are prototypes used for future requirements discovery.

Requirements Validation and Verification Matrices

A requirements validation matrix is an artifact that associates high-level requirements to certain system attributes for the purposes of tradeoff analysis and confirmation of requirement intent. Appropriate system attributes can include business need, safety, requirement volatility, and other factors. Feature cost could also be included in the matrix, but this issue is discussed in Chapter 10. One format for the requirements validation matrix for several requirements from the baggage handling system is shown in Table 5.2.

Table 5.2 Requirements Validation Matrix

Requirement Number	*Safety Impact (High = 10)*	*Volatility (High = 10)*	*Business Need (High = 10)*
3.1	10	2	10
3.2	2	1	5
…	…	…	…
3.210	3	6	7
3.211	5	10	1

Although the requirements validation matrix is used early in the project life cycle, a requirement verification matrix is used later. The requirements verification matrix associates requirements with test cases that verify that the requirement has been met. Such a matrix facilitates review of requirements and the tests, and provides an easy mechanism to track the status of test case design and implementation. An excerpt from a requirements verification matrix for the smart home system SRS documents is shown in Table 5.3.

Here the requirements forming the SRS are listed verbatim in the far left column, the tests that verify those requirements are listed in the center column, and the test case status is in the far right column. The status field can contain "passed," "failed," "not run," "omitted," or any variation of these keywords. Additional columns can be added to the matrix to indicate when the test was run, who conducted the test, where the test was run, the status of testing equipment used, and for comments and other relevant information.

The requirements verification matrix is easily made part of the master test plan and can be updated throughout the project to give a record of all requirements testing.

Table 5.3 A Sample Requirements Validation Matrix for the Smart Home System SRS in Appendix A

Requirement	*Test Cases*	*Status*
9.13.1 System shall provide wireless support for driving any number of wall-mounted monitors for picture display.	T-1711 T-1712 T-1715	Passed Passed Passed
9.13.2 System shall provide web-based interface for authenticated users to publish new photos for display on wall monitors.	T-1711 T-1715 T-1811	Passed Failed Passed
9.13.3 System shall allow users to configure which pictures get displayed.	T-1712 T-1715 T-1811 T-1812 T-1819	Passed Passed Passed Passed Not run
9.13.4 System shall allow users to configure which remote users can submit pictures to which wall monitor.	T-1712 T-1715 T-1716 T-1812	Passed Passed Passed Passed

Importance of Measurement in Requirements Validation and Verification

Imagine an argument involving the question of which boxer was better: Muhammad Ali or Joe Louis? Now, all kinds of elaborate simulation models have been contrived to settle this argument. You can try to argue superiority based on particular fighting characteristics. Some have modeled these characteristics to create simulated fights. Still others have looked at the fighting characteristics and results against similar opponents. You could even argue about training techniques and the quality of their managers, trainers, and promoters. But there is no conclusive evidence that one fighter was better than the other or that they were evenly matched because there are no direct metrics available to make just a value judgment. Now consider an argument involving who is the best long jumper of all time. The answer should be easy: Mike Powell, who holds the record of 8.95 meters (with no wind at his back). No one has ever jumped farther, and Powell broke Bob Beamon's record, which stood for 23 years. You can try to argue that Beamon was better in terms of other characteristics—competitive spirit, tenacity, resiliency, sportsmanship, and so on—but those are all unmeasurable.

Now imagine an argument with a customer involving whether a requirement that "the software shall be easy to use" was met. You contend that the requirement was met because, "Look, the software is easy to use." The customer disagrees because she feels that "The software is too hard to use." But you can't win the argument because you have no metrics. It is rather disappointing that as software engineers we are often no better off than two boxing pundits arguing from barstools (Laplante, Agresti, and Djavanshir 2007).

So which qualities should you consider and measure? Any collection of qualities is sometimes referred to as the "ilities." There are many possible qualities that comprise the "ilities," including:

- Accuracy
- Completeness
- Consistency
- Correctness
- Efficiency
- Expandability
- Interoperability
- Maintainability
- Manageability
- Portability
- Readability
- Reusability
- Reliability
- Safety
- Security
- Survivability
- Testability
- Understandability
- Usability

This is not an exhaustive list. In any case, for each requirement containing one of the "ilities" there needs to be an associated metric to determine if the requirement has been met. Requirements measurement is a concept that we revisit repeatedly.

Goal/Question/Metric Analysis

We previously mentioned the use of goal-based analysis for requirements verification and validation. But how can we generate the metrics that we need? The goal/question/metric (GQM) paradigm is an analysis technique that helps in the selection of an appropriate metric.

To use the technique, you follow three simple rules. First state the goals of the measurement; that is, "What is the organization trying to achieve?" Next, derive from each goal the questions that must be answered to determine if the goals are being met. Finally, decide what must be measured in order to be able to answer the questions (Basili, Caldiera, and Rombach 1994).

Here is an example of using GQM to define metrics that are "useful." Suppose that the stated goal for the system is, "The system shall be easy to use." It is hoped that you will agree that "easy to use" is impossible to measure objectively. So how do we approach its measurement? We do it by creating questions that help describe what "easy to use" means. For example, one question that fits this description is, "How many expert, intermediate, and novice users use the system?" The rationale for this question is that an easy-to-use system should be used by everyone. Now we need to know an appropriate metric to answer this question. Here is one way to obtain that metric: provide the system in an open lab for a period of time and measure the number and percentage of each user type who uses the system during that time. If a disproportionate number of users are expert, for example, then it may be concluded that the system is not easy to use. If an equal proportion of expert, intermediate, and novice users use the system, then it might be that the system is "easy to use."

Consider another question that addresses the goal of "easy to use." How long does it take a new user to master features 1 through 25 with only eight hours of training? The rationale is that certain minimum features needed to use the system adequately ought not to take too long to train. An associated metric for this question then is obtained by taking a random sample of novice users, giving them the same eight hours of training, and then testing the students to see if they can use features 1 through 25 to some minimum standard.

Following such a process to drive questions and associated metrics from goals is a good path to deriving measureable requirements and at the same time helping to refine and improve the quality of the requirements themselves.

Standards for Validation and Verification

There are various international standards for the processes and documentation involved in verification and validation of systems and software. Many of these have been sponsored or cosponsored by the Institute for Electrical and Electronics Engineers (IEEE).

Whatever requirements V&V techniques are used, a software requirements V&V plan should always be written to accompany any major or critical software application.

IEEE Std 1012-2012, IEEE Standard for Software Verification and Validation, provides some guidelines to help prepare verification and validation plans. Figure 5.2 shows the recommended V&V plan outline (IEEE 1012).

IEEE Standard 830

IEEE 830 is perhaps the most important of all the standards that relate to requirements engineering. IEEE 830 "describes recommended approaches for the specification of software requirements." The standard attempts to help the following:

1. Software customers to accurately describe what they wish to obtain
2. Software suppliers to understand exactly what the customer wants
3. Individuals to accomplish the following goals:
   a. Develop a standard software requirements specification (SRS) outline for their own organizations
   b. Define the format and content of their specific software requirements specifications
   c. Develop additional local supporting items, such as an SRS quality checklist or an SRS writer's handbook. (IEEE 830)

But from a risk mitigation standpoint, we are most interested in the qualities of goodness for requirements documents that are described. These are:

- Correct
- Unambiguous
- Complete
- Consistent
- Ranked for importance or stability
- Verifiable
- Modifiable
- Traceable

Let us describe each of these qualities in some detail.

Correctness

Correctness means that any requirement listed is one that needs to be met (i.e., incorrect requirements specifications specify unwanted behavior). Correctness is an important quality of an SRS document because unwanted behavior is, well, unwanted.

Sometimes unwanted behavior is not always obvious. Consider the following requirement for a computer security system.

1. Purpose
2. Referenced documents
3. Definitions
4. V&V overview
   - 4.1 Organization
   - 4.2 Master schedule
   - 4.3 Software integrity level scheme
   - 4.4 Resources summary
   - 4.5 Responsibilities
   - 4.6 Tools, techniques, and methods
5. V&V processes
   - 5.1 Process: Management
     - 5.1.1 Activity: Management of V&V
   - 5.2 Process: Acquisition
     - 5.2.1 Activity: Acquisition support V&V
   - 5.3 Process: Supply
     - 5.3.1 Activity: Planning V&V
   - 5.4 Process: Development
     - 5.4.1 Activity: Concept V&V
     - 5.4.2 Activity: Requirements V&V
     - 5.4.3 Activity: Design V&V
     - 5.4.4 Activity: Implementation V&V
     - 5.4.5 Activity: Test V&V
     - 5.4.6 Activity: Installation and checkout V&V
   - 5.5 Process: Operation
     - 5.5.1 Activity: Operation V&V
   - 5.6 Process: Maintenance
     - 5.6.1 Activity: Maintenance V&V
6. V&V reporting requirements
   - 6.1 Task reports
   - 6.2 Activity summary reports
   - 6.3 Anomaly reports
   - 6.4 V&V final report
   - 6.5 Special studies reports (optional)
   - 6.6 Other reports (optional)
7. V&V Administrative requirements
   - 7.1 Anomaly resolution and reporting
   - 7.2 Task iteration policy
   - 7.3 Deviation policy
   - 7.4 Control procedures
   - 7.5 Standards, practices, and conventions
8. V&V test documentation requirements

Figure 5.2 Recommended V&V plan table of contents. (Reprinted from IEEE Std 1012-2012, Institute for Electrical and Electronics Engineers, Piscataway, NJ, 2004. With permission.)

2.1.1 All passwords and user IDs shall be unique.

There is a problem with this requirement, for, if user A tries to set a password that is already taken by another user, the system has to indicate so. This gives user A knowledge of the password of another user, which could be used for an attack. Clearly this is incorrect behavior that we do not want to specify.

Various techniques can be used to study correctness including reviews and inspections, but none of these is perfect and it is probably the case that more than one review or inspection needs to be employed to ensure correctness.

Ambiguity

We define ambiguity by complementation: an SRS document is unambiguous if each specification element can have only one interpretation.

Here is an example of a specification document in which ambiguous behavior was described. In a certain automobile (belonging to the author) an indicator light is displayed (in the shape of an engine) when certain exceptional conditions occur. These conditions include poor fuel quality, fuel cap not tightened properly, and other fuel-related faults. According to the user's manual, if the cause of the problem is relatively minor, such as the fuel cap not tightened, the system will reset the light upon:

> . . . removing the exceptional condition followed by three consecutive error-free cold starts. A cold start is defined as a start up that has not been preceded by another engine start up in the last eight hours, followed by several minutes of either highway or city driving.*

Can you see the problem in this functional definition? Aside from its confusing wording, the requirement doesn't make sense. If you wait eight hours from the previous start up, then start the engine to drive somewhere, you have to wait at least eight hours to start up and drive back to your origin. If you have any warm start before three consecutive cold starts, the sequence has to begin again. Is the only possible way to satisfy this condition to drive somewhere, wait there for eight hours, and then drive back to the origin (three times in a row)? Or, drive around for a while, return to the origin, wait eight hours, then do it again (two times more)? This sequence of events is very hard to follow, and in fact, after one month of trying, the author could not get the light to reset without disconnecting and reconnecting the battery.

Another reason why ambiguity is so dangerous is that, in an ambiguous requirements specification, literal requirements satisfaction may be achieved but

* This is not a verbatim quote. It is a representation of the manual's verbiage to avoid exposing the identity of the vehicle.

not customer satisfaction. "I know that is what I said I wanted, but now that I see it in action, I realize that I really meant something else" is an unfortunate refrain. Or consider this fictitious quote, "Oh, you meant 'that' lever; I thought you meant the other one." We would never want this scenario to be played out late in the system's life cycle.

Some of the techniques that could be used to resolve ambiguity of SRS documents include formal reviews, viewpoint resolution, and formal modeling of the specification.

Completeness

Boehm (1984) notes that a requirement is complete "to the extent that all of its parts are present and each part is fully developed." This definition is corroborated by Menzel et al. (2010): "[Completeness is] the sense of whether the specification contains all requirements and whether all requirements contained in the specification are completely specified."

We use the following definition for completeness: an SRS document is complete if there is no missing functionality; that is, all appropriate desirable and undesirable behaviors are specified. Recall from Figure 1.3 that there is usually a mismatch between desired behaviors and specified behaviors: there is always some unspecified behavior, as well as undesirable behavior, that finds its way into the system that should be explicitly prohibited. Either case can lead to literal requirements satisfaction but not customer satisfaction.

Completeness is a difficult quality to prove. How do you know when something is missing? Typical techniques for reducing incompleteness include various reviews, viewpoint resolution, and the act of test case generation. Test-driven development, which is discussed in Chapter 7, has the effect of asking, "What's missing from here," and, "What can go wrong?" Answering these questions will tend to lead to systems features being uncovered. QFD is also a powerful technique to combat incompleteness because of the comparison of the system under consideration with competing systems. A competitive analysis is a fundamental component of QFD.

Consistency

The consistency of the SRS document can take two forms: internal consistency (i.e., satisfaction of one requirement does not preclude satisfaction of another) and external consistency (i.e., the SRS is in agreement with all other applicable documents and standards).

Here is a simple example of illustrating an internal inconsistency. The following are requirements for some kind of pneumatic control system, possibly relating to the baggage handling system.

3.1 If the lever is in position 1 then valve 1 is opened.
3.2 If the lever is in position 1 then valve 1 is closed.

Clearly Requirements 3.1 and 3.2 are inconsistent and therefore invalid; when the lever is in position 1 the valve cannot be both open and closed at the same time. More complex consistency checking is discussed in Chapter 6.

When either internal or external inconsistency is present in the SRS, it can lead to difficulties in meeting requirements and delays and frustration downstream. Internal and external consistency can be checked through reviews, viewpoint resolution, various formal methods, and prototyping.

Ranking

A requirements set is ranked if the items are prioritized for importance or stability. Importance is a relative term, and its meaning needs to be resolved on a case-by-case basis. Stability means the likelihood that the requirement would change. For example, a hospital information system will always have doctors, nurses, and patients (but governing legislation will change). The ranking could use a numerical scale (positive integers or real numbers), a simple rating system (e.g., mandatory, desirable, optional), or could be ranked by mission criticality.

For example, NASA uses a four-level ranking system. Level 1 requirements are mission-level requirements that are very high level and very rarely change. Level 2 requirements are high level with minimal change. Level 3 requirements are those requirements that can be derived from level 2 requirements. That is, each level 2 requirement traces to one or more level 3 requirements. Contracts usually bid at this level of detail. Finally, at level 4 are detailed requirements that are typically used to design and code the system (Wilson et al. 1997).

Ranking is an extremely important quality of an SRS document. Suppose in the course of system design two requirements cannot be met simultaneously. It becomes easier to decide which requirement to relax based on its ranking. In addition to being useful for tradeoff engineering, ranking can be used for cost estimation and negotiation, and for dispute resolution. Ranking validation is easy enough through reviews and viewpoint resolution (to agree upon the rankings).

Verifiability

An SRS is verifiable if satisfaction of each requirement can be established using measurement or some other unambiguous means. This quality is important because a requirement that cannot be shown to be met has not been met. When requirements cannot be measured, they cannot be met and disputes will follow.

Verifiability can be explored through various reviews, through test case design (design-driven development), and through viewpoint resolution.

Modifiability

Modifiability means that the SRS and structure of the document will readily yield to changes. Usually this means that the document is numbered, stored in a convenient electronic format, and compatible with common document processing and configuration tools.

It is obvious why modifiability is an important quality of an SRS document: requirements will change. Ease of modification will also reduce costs, assist in meeting schedules, and facilitate communications. Reviews and inspections are the most obvious way to assess a document's modifiability.

Traceability

An SRS is traceable if each requirement is clearly identifiable, and all linkages to other requirements (e.g., dependencies) are clearly marked. Traceability is an essential quality for effective communications about requirements, to facilitate easy modification, and even for legal considerations. For example, in the case of a dispute, it is helpful to show that responsible linking of related requirements was done.

In addition, each requirement should have a link to at least one other requirement. Traceability can be measured using networklike analyses. For example, we could count the efferent (inward) and afferent (outward) coupling as indicated by the key phrases "uses," "is referenced by," "references," "is used by," and so on.

Generally, we would like each requirement to be tested by more than one test case. At the same time, we would like each test case to exercise more than one requirement. The "test span" metrics are used to characterize the test plan and identify insufficient or excessive testing:

- Requirements per test
- Tests per requirement

Research is still ongoing to determine appropriate statistics for these metrics. But at the very least, you can use these metrics to look for inconsistencies and nonuniform test coverage. Of course, there is always a tradeoff between time and cost of testing versus the comprehensiveness of testing. But testing is not the subject of this book.

Group reviews and inspections and automated tools can also be used to check for traceability between requirements to or from tests. We investigate traceability further in Chapter 8.

Example Validation of Requirements

Consider Requirements 8.1.1 through 8.1.18 for the smart home from Appendix A. Now let's list the IEEE 830 criteria and evaluate the requirements along these criteria.

Correct

The requirements appear to be consistent with common understanding of television recording works in a home environment. But it would be hard to be certain if this set is correct without further analysis using techniques such as using focus groups and experts.

Unambiguous

There are quite a few unclear or ambiguous phrases in these requirements that are revealed by asking, "How do I test this?" These problems will need to be resolved. For example, 8.1.1 says that the system shall record "any show on television." There are many problems with this statement: any show on US television or around the world? What about pay-per-view, and so on? Requirement 8.1.4 says the system shall "make storage for recorded shows expandable." What does making storage mean (allocation)? How much expansion should be allowed? Requirement 8.1.6 talks about searching; but based on what: program name, time it was recorded, subject? Requirement 8.1.10 describes selecting "quality for recording." What does "quality" mean? Requirement 8.1.13 mentions storing "x episodes." Is "x" a natural number, an integer, a complex number, a quaternion? There are other ambiguities here that could bear improvement.

Complete

Although there are no TBD requirements in the document, we can't really say if this set of requirements is incomplete without using such requirements validation techniques as QFD or focus groups.

Consistent

Without conducting a formal consistency analysis, we cannot be sure. For this segment of requirements, however, it probably wouldn't be too hard inasmuch as the "shall" statements are relatively straightforward. Therefore, although there would be a large number of logical variables, there would not be many compound propositions to evaluate (this approach is discussed in Chapter 6). You can use automatic consistency checkers to validate this segment of requirements, or you could actually use a spreadsheet program to do the job. In this situation, however, an informal reading is probably adequate. An informal reading of the requirements does not reveal any obvious inconsistencies.

Ranked for Importance or Stability

We know from reading the opening paragraphs of Section 8 of the SRS that this set of requirements is ranked as "medium" in importance. Ranking a large cluster

of requirements as having the same priority does not particularly help except, however, when the need arises to sacrifice some requirements in order to meet budget or schedule. It would be better if each minor requirement were ranked separately so that appropriate sacrifices can be made.

Verifiable

Because of the aforementioned ambiguities, these requirements are not fully verifiable.

Modifiable

There are no obvious challenges to modifying and maintaining these requirements. Using an appropriate change management tool to track the changes, their sources, and dates would be needed.

Traceable

The requirements are numbered in a way that appears to be consistent. There are no internal references or usages of one requirement to another in this set.

NASA Requirements Testing

One would think that the American space agency NASA is a place where rigorous requirements engineering is conducted. This is a correct assumption. Given that NASA is engaged in the engineering of very high-profile, high-cost, and most important, life-critical systems, the techniques used and developed here are state of the art. Table 5.4 contains an excerpt from NASA Procedural Requirements for requirements engineering (which is now obsolete but still useful for our purposes).

NASA is heavily invested in the use of formal methods for requirements verification and validation, and a number of techniques and tools for this purpose have been developed.

Notice that verification matrices are specifically mentioned as helping to accomplish the software requirements engineering goals. Requirements management, discussed in Chapter 9, is specifically mentioned in the directive.

NASA ARM Tool

The NASA Automated Requirements Measurement (ARM) Tool was developed at NASA's Software Assurance Technology Center at the Goddard Space Flight Center in Greenbelt, Maryland. This tool conducts an analysis of the text of the SRS document and reports certain metrics. The metrics are divided into two categories: micro- and macrolevel metrics. Microlevel indicators count the

Table 5.4 Excerpt from NASA Procedural Requirements for Requirements Engineering

3.1.1 Requirements Development
3.1.1.1 The project shall document the software requirements. [SWE-049]
Note: The requirements for the content of each Software Requirement Specification and Data Dictionary are defined in Chapter 5.
3.1.1.2 The project shall identify, develop, document, approve, and maintain software requirements based on analysis of customer and other stakeholder requirements and the operational concepts. [SWE-050]
3.1.1.3 The project shall perform software requirements analysis based on flowed-down and derived requirements from the top-level systems engineering requirements and the hardware specifications and design. [SWE-051]
Note: This analysis is for safety criticality, correctness, consistency, clarity, completeness, traceability, feasibility, verifiability, and maintainability. This includes the allocation of functional and performance requirements to functions and subfunctions.
3.1.1.4 The project shall perform, document, and maintain bi-directional traceability between the software requirement and the higher level requirement. [SWE-052]
Note: The project should identify any orphaned or widowed requirements (no parent or no child) associated with reused software.
3.1.2 Requirements Management
3.1.2.1 The project shall collect and manage changes to the software requirements. [SWE-053]
Note: The project should analyze and document changes to requirements for cost, technical, and schedule impacts.
3.1.2.2 The project shall identify inconsistencies between requirements, project plans, and software products and initiate corrective actions. [SWE-054]
Note: A verification matrix supports the accomplishment of this requirement.

Source: NASA Procedural Requirements, 2004. NASA Software Engineering Requirements, NPR 7150.2. With permission.

occurrences of specific keyword types. Macrolevel indicators are coarse-grained metrics of the SRS documentation.*

* NASA ceased supporting ARM and a successor tool, e-Smart, sometime around 2009. The author's students are reconstructing ARM, in several variations, for release. If you would like to access this tool, please contact the author at plaplante@psu.edu for more information.

Microlevel indicators include:

- Imperatives
- Continuances
- Directives
- Options
- Weak phrases

Macrolevel indicators include:

- Size of requirements
- Text structure
- Specification depth
- Readability

These micro- and macrolevel indicators are described in some detail.

In addition, various ratios can be computed using macro- and microlevel indicators. No particular thresholds for the metrics are given (research is still being conducted in this regard). However, at the end of this section, summary metrics for 56 NASA projects are given for comparison.

A description of the metrics and some excerpts from the ARM report for creating the smart home SRS document can be found in Appendix A. The definitions are derived from those reported by the tool and described by the authors of the tool in a related report (Hammer et al. 1998).

Imperatives

The first metric, imperatives, is a microindicator that counts the words and phrases that command something must be provided. Imperatives include "shall," "will," "must," and others as described in Table 4.2.

A more precise specification will have a high number of "shall" or "must" imperatives relative to other imperative types. Note that the word "should" is not recommended for use in an SRS. From both a logical and legal point of view "should" places too much discretion in the hands of system designers. The counts of imperatives found in the smart home SRS document are shown in Table 5.5.

An excerpt of the imperatives capture from the ARM output is shown in Figure 5.3.

Continuances

Continuances are phrases that follow an imperative and precede the definition of lower-level requirement specifications. Continuances indicate that requirements have been organized and structured. Examples and counts of continuances found in the smart home SRS document are shown in Table 5.6. The symbol ":" is treated as a continuance when it follows an imperative and precedes a requirement definition.

Table 5.5 ARM Counts of Imperatives Found in the Smart Home SRS Document in Appendix A

Imperatives	*Occurrences*
shall	308
must	0
is required to	0
are applicable	0
are to	1
responsible for	0
will	51
should	7
Total	367

These characteristics contribute to the ease with which the requirement specification document can be changed. Too many continuances, however, indicate multiple complex requirements that may not be adequately reflected in resource and schedule estimates.

Directives

The microindicator "directives" count those words or phrases that indicate the document contains examples or other illustrative information. Directives point to information that makes the specified requirements more understandable. Typical directives and their counts found in the smart home SRS document are shown in Table 5.7. Generally, the higher the number of total directives is, the more precisely the requirements are defined.

Options

Options are those words that give the developer latitude in satisfying the specifications. At the same time, options give less control to the customer. Options and their counts found in the smart home SRS document are shown in Table 5.8.

Weak Phrases

Weak phrases are clauses that are subject to multiple interpretations and uncertainty, and therefore can lead to requirements errors. Use of phrases such as "adequate" and "as appropriate" indicate that what is required is either defined elsewhere or

shall # 1: In Line No. 169, ParNo. 3.1.1, @ Depth 3
3.1.1 System SHALL operate on a system capable of multi-processing.

will # 51: In Line No. 6224, ParNo. 9.12.3, @ Depth 3
9.12.3 System SHALL allow users to record greeting message that WILL be played after user defined number of rings.

should # 7: I Line No. 557, ParNo. 9.8, @ Depth 2
In the future this SHOULD be extended such that any commands can be programmed to control any device or system interfaced by the SH.

be able to # 1: In Line No. 324, ParNo. 7.3, @ Depth 2
Occupants and users of the SH's system should BE ABLE TO monitor the home from anywhere they wish.

normal #1: In Line No. 490, ParNo. 9.3.14, @ Depth 3
9.3.14 Hot tub cover shall close with button press or if no activity / motion is detected for some time range, and water displacement levels are NORMAL (no one in the tub).

provide for # 1: In Line No. 104, ParNo. 2., @ Depth 2
The summation and harmonization of all the six categories of the SH will PROVIDE FOR a truly rewarding living experience for the occupants of the SH.

easy to # 1: In Line No. 193, ParNo. 4.1.2, @ Depth 3
4.1.2 System shall be EASY TO use.

can # 1: In Line No. 124, ParNo. 2., @ Depth 2
Ensuring the existing structure CAN support the improvements

may # 1: In Line No. 180, ParNo. 3.1.9, @ Depth 3
3.1.9 System MAY contain separate SAN device for storage flexibility

Figure 5.3 Excerpt of ARM output for smart home requirements specification document.

worse, the requirement is open to subjective interpretation. Phrases such as "but not limited to" and "as a minimum" provide the basis for expanding requirements that have been identified or for adding future requirements. The counts of weak phrases for the smart home SRS document are shown in Table 5.9.

The total number of weak phrases is an important metric that indicates the extent to which the specification is ambiguous and incomplete.

Incomplete

The "incomplete" microindicator counts words that imply something is missing in the document, for whatever reason (e.g., future expansion, undetermined requirements). The most common incomplete notation is "TBD" for "to be determined."

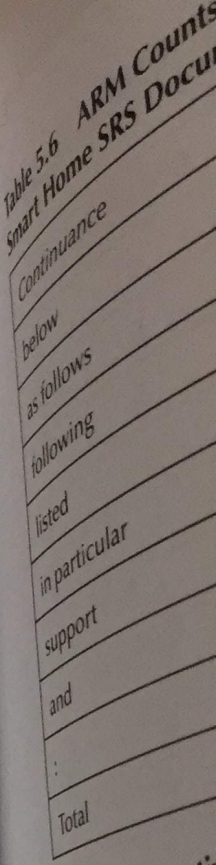

Table 5.6 ARM Counts for Continuances in the Smart Home SRS Document in Appendix A

Continuance	*Occurrence*
below	0
as follows	0
following	0
listed	0
in particular	0
support	0
and	85
:	2
Total	87

Table 5.7 Directives Found in the Smart Home SRS Document in Appendix A

Directive	*Occurrence*
e.g.	0
i.e.	14
For example	0
Figure	0
Table	0
Note	0
Total	14

Table 5.8 Options and Their Counts Found in the Smart Home SRS Document in Appendix A

Option Phrases	*Occurrence*
can	7
may	23
optionally	0
Total	30

Table 5.9 Weak Phrases for the Smart Home SRS Document in Appendix A

Weak Phrase	*Occurrence*
adequate	0
as appropriate	0
be able to	3
be capable of	0
capability of	0
capability to	0
effective	0
as required	0
normal	1
provide for	1
timely	0
easy to	1
Total	6

Variations of "TBD" include:

- TBD: "to be determined"
- TBS: "to be scheduled"
- TBE: "to be established" or "yet to be estimated"
- TBC: "to be computed"
- TBR: "to be resolved"
- "Not defined" and "not determined": Explicitly state that a specification statement is incomplete
- "But not limited to" and "as a minimum": Phrases that permit modifications or additions to the specification

Incomplete words and phrases found in the smart home SRS document are shown in Table 5.10.

Leaving incompleteness in the SRS document is an invitation to disaster later in the project. Although it is likely that there may be a few incomplete terms in a well-written SRS due to pending requirements, the number of such words should be kept to an absolute minimum.

Table 5.10 Incomplete Words and Phrases Found in the Smart Home SRS Document in Appendix A

Incomplete Term	*Occurrence*
TBD	0
TBS	0
TBE	0
TBC	0
TBR	0
not defined	0
not determined	0
but not limited to	0
as a minimum	0
Total	0

Subjects

Subjects are a count of unique combinations of words immediately preceding imperatives in the source file. This count is an indication of the scope of subjects addressed by the specification.

The ARM tool counted a total of 372 subjects in the smart home SRS document in Appendix A.

Specification Depth

Specification depth counts the number of imperatives at each level of the document and reflects the structure of the requirements. The topological structure of requirements was discussed in Chapter 4. The numbering and specification structural counts for the smart home SRS document as computed by the NASA ARM tool are provided in Table 5.11.

These counts indicate that the SRS requirements hierarchy has a "diamond" shape in the manner of Figure 4.8c.

Readability Statistics

In Chapter 4 we discussed the importance of clarity in the SRS document. There are various ways to evaluate reading levels, but most techniques use some formulation of characters or syllables per words and words per sentence. For example,

Table 5.11 Numbering and Specification Structure Statistics for Smart Home SRS Document in Appendix A

Numbering Structure		Specification Structure	
Depth	Occurrence	Depth	Occurrence
1	19	1	0
2	71	2	50
3	265	3	258
4	65	4	64
5	0	5	0
6	0	6	0
7	0	7	0
8	0	8	0
9	0	9	0
Total	420	**Total**	372

the Flesch Reading Ease index is based on the average number of syllables per word and of words per sentence. Standard writing tends to fall in the 60–70 range but apparently, a higher score increases readability.

The Flesch-Kincaid Grade Level index is supposed to reflect a grade-school writing level, so a score of 12 means that someone with a 12th grade education would understand the writing. But standard writing averages 7th to 8th grade, and a much higher score is not necessarily good; higher-level writing would be harder to understand. The Flesch-Kincaid Grade Level indicator is also based on the average number of syllables per word and on words per sentence. There are other grade-level indicators as well (Wilson, Rosenberg, and Hyatt 1997).

The NASA ARM tool does not provide the ability to count these metrics, but most versions of the popular Microsoft Word can provide at least some relevant statistics. For example, the version of Word 2007 used to prepare this manuscript will compute various word, character, paragraph, and sentence counts and averages. It will also compute the Flesch Reading Ease and Flesch-Kincaid Grade Level metrics (consult the user's manual or online help feature to determine how to compute such metrics for your word processor, if available). In any case, we used Word to calculate the statistics for the smart home SRS document and obtained the output shown in Figure 5.4.

The SRS document is assessed to be at a 12th grade reading level. The low number of sentences per paragraph (1.2) is an artifact of the way the tool counted each numbered requirement as a new paragraph.

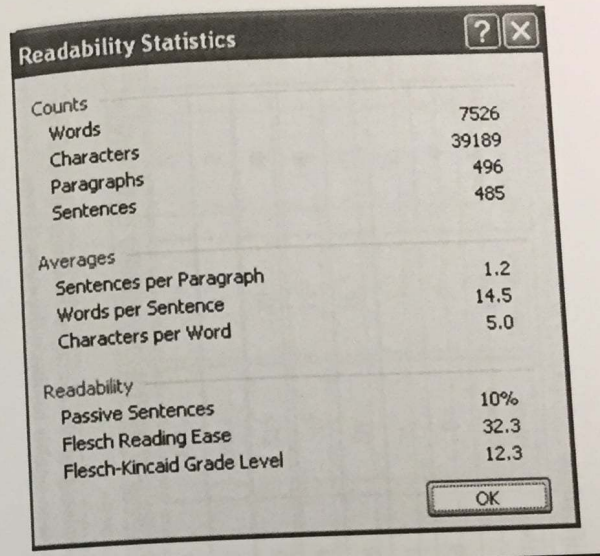

Figure 5.4 Readability statistics for smart home SRS document.

Summary of NASA Metrics

To get some sense of proportion and relevance to the ARM indicators, Wilson and colleagues studied 56 NASA software systems ranging in size from 143 to 4,772 lines of code (Wilson, Rosenberg, and Hyatt 1997). They used their tool to collect statistics about these systems, which are summarized in Table 5.12.

From Table 5.12 we notice that one specification was only 143 lines of text and that the longest was 28,000 lines of text. It is interesting to note from Table 5.12 that even NASA specifications have "TBDs" and often, many options. Validatable and verifiable are two additional properties in the table (Wilson et al. 1997).

Aside from the obvious interpretations, why are these ARM indicators important? Table 5.13 gives us the intriguing answer: because there is a correlation between these indicators and the IEEE 830 qualities that we have already discussed.

Looking at the table it seems that the text structure and depth are indicators of internal consistency (but not external). The number of directives and weak phrases are correlated (positively and negatively, respectively) with correctness. All of the microindicators are linked to testability (both in a positive and negative correlation: you need "just enough" directives but not too many). Finally, all quality indicators (except for readability) contribute to completeness.

Table 5.12 Sample Statistics from 56 NASA Requirements Specifications

	Lines of Text	*Imperatives*	*Continuances*	*Directives*	*Weak Phrases*	*TBD, TBS, TBR*	*Option (can, may…)*
Minimum	143	25	15	0	0	0	0
Median	2,265	382	183	21	37	7	27
Average	4,772	682	423	49	70	25	63
Max	28,459	3,896	118[a]	224	4[a]	32	130
Std Dev	759	156	99	12	21	20	39
Level 3 Specs	1,011	588	577	10	242	1	5
Level 4 Specs	1,432	917	289	9	393	2	2

Source: W.M. Wilson, L.H. Rosenberg, and L.E. Hyatt (1997). In *Proceedings of the 19th International Conference on Software Engineering*, 161–171.

[a] These two figures are clearly wrong. How can the maximum numbers be less than the averages? However, all instances of this table in the literature show the same error and the author has been unable to obtain the correct ones.

Table 5.13 Cross Reference of NASA Indicators to IEEE 830 Qualities

	Indicators of Quality Attributes										
	Quality Attributes										
Categories of Quality Indicators	*Complete*	*Consistent*	*Correct*	*Modifiable*	*Ranked*	*Testable*	*Traceable*	*Unambiguous*	*Understandable*	*Validatable*	*Verifiable*
Imperatives	x			x			x	x	x	x	x
Continuances	x			x	x	x	x	x	x	x	x
Directives	x		x			x		x	x	x	x
Options	x					x		x	x	x	
Weak phrases	x		x			x		x	x	x	x
Size	x					x		x	x	x	x
Text structure	x	x		x	x		x		x		x
Specification depth	x	x		x			x		x		x
Readability				x		x	x	x	x	x	x

Source: W.M. Wilson, L.H. Rosenberg, and L.E. Hyatt (1997). In *Proceedings of the 19th International Conference on Software Engineering*, 161–171.

Exercises

5.1 What can be some pitfalls to watch out for in ranking requirements?
5.2 Describe two different ways to identify ambiguity in an SRS.
5.3 Which of the IEEE Standard 830 qualities seem most important? Can you rank these?
5.4 For an available SRS document, conduct an informal assessment of its IEEE 830 qualities.
5.5 For each Quality Attribute in Table 5.12 discuss its relationship to the Categories of Quality indicators.
5.6 Should implementation risk be discussed with customers?
5.7 What are the advantages and risks of having requirements engineering conducted (or assisted) by an outside firm or consultants?
5.8 Create a traceability matrix for the SRS in
   a. Appendix A
   b. Appendix B
5.9 Calculate the Requirements per Test and Tests per Requirements metrics for the data shown in Table 5.2. Do you see any inconsistencies?
5.10 Consider the requirements in the SRS of Appendix A.
   a. Which of these could be improved through the use of visual formalisms such as various UML diagrams?
   b. Select three of these and create the visual formalism.
5.11 The requirements in Appendix B are not ranked. Using reasonable assumptions, rank these requirements. (Use a requirements ranking matrix for brevity.)
5.12 Why is the NASA ARM tool useful in addition to objective techniques of SRS risk mitigation?
5.13 Install and run the NASA ARM tool on any available SRS document (contact the author for more information on obtaining an executable version). What can you infer from the results?
*5.14 Reconstruct the ARM tool in your favorite programming language. Note that the generation of the statistics is relatively simple. The challenge is in parsing the SRS document. Place restrictions on the input to the tool as needed.

References

Bahill, T.A. and Henderson, S.J. (2005). Requirements development, verification, and validation exhibited in famous failures, *Systems Engineering*, 8(1): 1–14.

Basili, V., Caldiera, G., and Rombach, H.D. (1994). Goal question metric approach. In *Encyclopedia of Software Engineering*. New York: John Wiley & Sons, pp. 528–532.

* This exercise is suitable for a small research assignment.