Workaday software requirements specifications

Robert M. Poston, Software Standards Editor

Recently, in this magazine and in other software journals, we have seen articles and discussions about formalism in specifications, understanding IEEE Std 830, and writing requirements for testability. These subjects are lofty-sounding to some, which may cause the articles to be brushed aside. People who barely have enough time to finish their basic tasks at work don’t have much energy left to start delving into the seemingly complex concepts embodied in such subjects.

All the subjects mentioned address improving software requirements specifications. Let’s look at three easy ways we can make an SRS more effective, according to the recent literature.

**Natural language.** Use the language naturally spoken by the end user. If the end user speaks English every day, use English. Limit the vocabulary to what is familiar to the end user. If the end user is a lawyer, use terms from the legal vernacular.

Natural languages pose a problem in that they are ambiguous. That is, words or arrangements of words can have many interpretations. So, we have to minimize ambiguities by following the rules in Figure 1 and by playing to the strong point of natural language—familiarity.

**Formal language.** Use formal specification languages to specify items and relationships that require mathematical preciseness. For example, if a product must perform algebraic computations, use algebra. If a product must operate on sets of items, use set and logic notations. If a product must manipulate data structures, use a data structure notation such as that illustrated in Figure 2. A software development team working on a large project will probably consider a formal specification method such as SADT, SREM, or PSL/PSA.

**Standard specifications.** Use a standard specification format to ensure that all information in the SRS is where it is supposed to be (very handy when you have to look up something). An outline from IEEE Std 830-1984 intended to guide the writing of an SRS is provided in Figure 3.

Now that we have reviewed the three suggestions for improving our SRS writing, let’s see how they work when applied to a simple problem like the one for the text formatter described in Figure 4.

---

**Standards bulletin board**

**IEEE P1003 OS Kernel Working Group report**

About 40 people attended the most recent meeting of the Operating Systems Kernel Working Group, June 9-11 in Portland, Oregon. Jim Isaak reported the following developments.

**Meetings.** NBS is coordinating the September 10-12 meeting in the Washington DC area; a host is being sought for lunches. The January 13-15 meeting in Denver, Colorado, will focus on ballot resolution. An April 16-18 meeting is tentatively planned for Europe, pending feedback from European participants on date and location and from US participants on expected attendance.

**Overlap.** A substantial review of overlap with the X3J11 group has been initiated. A subcommittee will recommend which routines should be dropped from the P1003 document, with referral to X3J11 materials, and what should be defined in the P1003 document. The objective is to have the two documents complement each other, with P1003 providing information designated in X3J11 as “implementation defined” and other specific characteristics.

**Interface.** Nine of 10 existing differences from the System V Interface Document have been resolved. The exception is use of the error message “EFAULT” for “time” and “wait.”

The working group decided that the error message, which reports addresses that are out of the process’s address space, should be standardized.

**Locking.** Enforcement locking for files and records has been moved to an appendix, and a subcommittee is being formed to deal with DBMS support issues. Advisory locking has been retained in the draft.

**Unix.** A Unix standard newsgroup is being formed and will be moderated by John Quarterman (mod. std. unix).

The group recommended that the current proposals and draft be made accessible on line as well; this will require some cross checks to make sure no policy problems exist.

**WG P1059, Software Verification and Validation, formed**

The IEEE Software Engineering Standards Subcommittee has begun an effort to develop a guide for software verification and validation. The guide will describe alternative approaches to good practice in the implementation of SV&V. The guide will identify and describe a number of activities, tools, techniques, and methodologies that have proved successful in past SV&V efforts of varying size, application, and complexity.

This effort is expected to take approximately three years. The working group is planning to meet quarterly at different geographic locations throughout the country.

Individuals interested in participating in this effort should contact either Jerry Mersky at Logicon, Inc., 1555 Wilson Blvd., Ste. 601, Arlington, VA 22209, (703) 525-2484, or Roger Blais at Lopez Road Facility, Dynamics Research Corporation, 60 Concord St., Wilmington, MA 01887, (617) 658-7685.
If we assume our end user is a novelist, we have to describe the text formatter in the natural language of a writer. The outline of IEEE Std 830 tells us we should itemize design constraints or describe the environment in which the product will run. For this description, we will specify the Unix environment and limit this discussion just to the technical requirements of the text formatter (Section 3.1 of "Specific Requirements" in Figure 3).

Section 3.1 contains the results of applying these three suggestions to the sample problem. The information boils down to what is contained in an input data dictionary (Figure 5), processing requirements (Figure 6), and an output data dictionary (Figure 7).

Requirements that are written as illustrated in these figures have three unique features. First, the result of every verb is an action visible outside the system. An example is R12, "eliminate any optional breaks." The optional breaks (leading or trailing any text characters) can be seen going into the text formatter, and the output can be seen to verify that the breaks were removed.

Second, every noun in the input and output data dictionaries represents only items visible outside the text formatter. The nouns correspond to external world entitles. Verbs and nouns must be visible outside the system for design information to be excluded from the SRS.

Third, from a tester's point of view, having the requirements itemized and clearly visible outside the system is especially valuable. The SRS identifies exactly what must be tested with what input to produce what expected output. We see that the SRS is organized for testability as well as for user understandability.

The academic community continues to explore and present ways to express software specification requirements. The techniques and terminology that catch on with the software development community eventually become standards of our industry.

---

**Rule 1. Use a standard format, IEEE Std. 830-1984.**

Reasons: To establish consistency between documents, to lower the probability of "I did not know where to store (or find) the information" errors, and to establish a checklist for good requirements characteristics.

**Rule 2. Use models for functional relationships, data flow, structure, and performance.**

Reasons: To express requirements more precisely and completely.

**Rule 3. Limit the structure of paragraphs to a list of individual sentences.**

Reasons: To increase the traceability and modifiability of the requirements, and to increase the ease of checking for completeness.

**Rule 4. Limit the structure of each sentence to a simple sentence with noncompounded verbs or nouns.**

Reasons: To increase the verifiability (testability) of each requirement.

**Rule 5. Limit the verbs and nouns in the sentences to a small set with a single specified definition for each word.**

Reasons: To improve consistency and to reduce ambiguity.

**Rule 6. Limit the verbs and nouns to terms that are the natural terms of the end user.**

Reasons: To increase user understanding of the requirements.

**Rule 7. Limit the verbs and nouns to actions and objects that are visible external to the product.**

Reasons: To reduce the amount of design data that goes into the requirements and to increase the testability of the product.

---

**Notation | Read as | For**
---|---|---
\( = \) | "is composed of" | Sequence
\( + \) | "and" | Selection
\( [\ ] \) | "select one of" | Repetition
\( m\{\ \} n \) | "iterated at least \( m \) times, but not more than \( n \) times" | "optional"
\( (\ ) \) | "optional" | "comment"
Given are a nonnegative integer MAXPOS and a character set including two "break characters" blank and new_line.
The program shall accept as input a finite sequence of characters and produce as output a sequence of characters satisfying the following conditions:
- it only differs from the input by having a single break character wherever the input has one or more break characters;
- any MAXPOS+ consecutive characters include a new_line;
- the number of new_line characters is minimal.
If (and only if) an input sequence contains a group of MAXPOS+1 consecutive nonbreak characters, there exists no such output. In this case, the program shall produce the output associated with the initial part of the sequence, up to and including the MAXPOS-th character of the first group, and report the error.

Figure 4. Requirements for a text formatter—Version 3.

| MAXPOS | = primitive data type = integer
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>acceptable values are from 1 to 132, inclusive default value = 80</td>
</tr>
</tbody>
</table>

I-STREAM = structure
- (BREAK) + WORD + 0(BREAK + WORD)n + (BREAK) + <EDT>
- shortest accepted structure = WORD + <EDT>
- longest accepted structure = a total of 5,000,000 CHARACTERS

WORD = structure
- 1[ WORD-CHARACTER ]value(MAXPOS)

WORD-CHARACTER = primitive data type = CHARACTER
- acceptable values are limited to the CHARACTERS with ASCII values between 033 and 126 inclusive.

BREAK = structure
- 1 [ BREAK-CHARACTER ]value(B_LIMIT)

BREAK-CHARACTER = primitive data type = enumerated
- acceptable values = [ <HT> | <SP> | <NL> ]

B_LIMIT = primitive data type = integer
- acceptable values are from 1 to 5,000,000, inclusive default value = 5,000,000

<HT> = primitive, ASCII value = 033 (TAB CHARACTER)

<SP> = primitive, ASCII value = 009 (SPACE CHARACTER)

<NL> = primitive, ASCII value = 010 (NEW-LINE CHARACTER)

<EDT> = primitive, End-Of-Text Delimiter, provided by the operating system

CHARACTER = ASCII standard character set codes.
- Values are between 033 and 126 inclusive, plus 009, 010, and 032.
- All other character values are trapped or filtered by the operating system and will not be input to textform.

Figure 5. Input data dictionary.

Figure 6. Processing requirements.

| COMPLETION CODE | = primitive data type = enumerated
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>acceptable values = [ &lt;1&gt;</td>
</tr>
<tr>
<td></td>
<td>Value               Meaning</td>
</tr>
<tr>
<td>&lt;1&gt;</td>
<td>normal completion</td>
</tr>
<tr>
<td>&lt;2&gt;</td>
<td>invalid MAXPOS</td>
</tr>
<tr>
<td>&lt;3&gt;</td>
<td>I-STREAM not present</td>
</tr>
<tr>
<td>&lt;4&gt;</td>
<td>empty I-STREAM ( &lt;EDT&gt; only)</td>
</tr>
<tr>
<td>&lt;5&gt;</td>
<td>oversize WORD</td>
</tr>
<tr>
<td>&lt;6&gt;</td>
<td>oversize BREAK</td>
</tr>
</tbody>
</table>

O-STREAM = structure
- 1 *[ LINE + <NL> ]

LINE = structure
- = WORD + 0( <SP> + WORD )

WORD see WORD in INPUT DATA DICTIONARY

<SP> see <SP> in INPUT DATA DICTIONARY

<NL> see <NL> in INPUT DATA DICTIONARY

Figure 7. Output data dictionary.