Preventing most-probable errors in requirements

Robert M. Poston,
Software Standards Editor

As inventive and receptive to change as most software development people are, they often are too pressed for time to try new ideas on today’s projects. Because good products start with good requirements, development professionals look first for ways to improve the quality of software requirements. However, job pressures intrude on these improvement attempts.

This is the second in a series on using standards to prevent most-probable errors in software development.

Consider developers who must hurry through requirements to appease managers pushing for code now. Project managers get their share of pressure—often to meet unrealistic, customer-demanded deadlines. SQA managers are supposed to assure product quality. But what a feat that is when the people developing the product and managing the project have little time or money to invest in writing better requirements.

A most-probable errors list provides affordable help for all these people in their quest for requirements quality. The July issue’s Software Standards (pp. 86-88) tells how to develop and use the list.

In 1984, the IEEE published a standard (ANSI/IEEE 830-1984) that guides the development of software requirements specifications. With that, specification writers on all kinds of software projects could adhere to a widely accepted set of criteria.

One after the other, software requirements specifications began to look like their predecessors in information content and format. Standardization was making repeatability possible, thus allowing more accurate project-to-project measurements and comparisons. A most-probable error spotted in requirements specification A could be listed and prevented in requirements specification B.

Investment and return. Creating a most-probable errors list requires an investment. One person from a project must be willing to spend four to eight hours compiling a most-probable errors list for requirements specifications, and every colleague on the project must be willing to spend as much as two hours reviewing and approving the list.

Usually, the peer review can be accomplished in one meeting.

What is the return from such a simple investment? It is the increased probability that software requirements will be written with fewer errors. Everyone on the project team now knows what is most likely to go wrong when writing requirements specifications. Forewarned is forearmed. Also, every individual now knows that his peers know what errors he is most likely to make. Peer awareness starts to work to the team’s advantage.

Very little time and effort has been invested, but something solid has been produced: a most-probable errors list accompanied by approval signatures from everybody on the project. Management will see that people on this project have taken decisive steps to improve quality by searching for those mistakes that caused the most trouble in the past and by making a commitment to avoid those mistakes in the future.

So there you have the minimum investment and payback associated with the most-probable errors list. Many people find the payback from this minimum, initial effort so encouraging that they want to invest further.

More comprehensive quality efforts involve the use of standards and reviews. A sample second-level investment plan or quality improvement plan might be outlined as follows:

1. The software development team meets for as many as four hours to critique a standard for a software requirements specification. (Std 830 is widely used.) At the end of this meeting, team members know what their requirements specification should look like and what it should contain.
2. The team develops a most-probable errors list. From the list, team members know what pitfalls to watch for.
3. The team writes a requirements specification.
4. The team reviews the requirements specification using the walkthrough or inspection technique (see M.E. Fagan, “Design and Code Inspections to Reduce Errors in Program Development,” IBM Systems J., No. 15, 1976, pp. 182-211). If you use a standard and the most-probable errors list, many errors will have been prevented. Of those that did slip through, many will be caught in the review.

“Surely such a simple process can’t be beneficial,” some say. But, if you don’t have much to invest, it certainly is worth a try. Stated another way, if you cannot afford the overhead of formal inspections, put the most-probable errors list and informal reviews to the test.

One company’s efforts. Software professionals at Digital Switch Corp. in Plano, Texas, went a step further with their most-probable errors list for requirements specifications. They took each error on their list and specified a technique to prevent that error from occurring again. That kind of thoroughness is more costly than the investments we’ve talked about. However, Digital Switch believes its return will be proportional to its investment.

Table 1 shows Digital Switch’s most-probable errors list. The people there said they tried to rank the items but ended up thinking that all items were equally important. To make it more widely applicable, the list has been generalized.
Table 1.
Most-probable errors in requirements (as compiled by Digital Switch Corp.).

<table>
<thead>
<tr>
<th>Category</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing information</td>
<td>Function or feature, Interface definition, Performance requirements, Response time, Database, input, output loads, Environment constraints, Quality requirements, Human factors, Reliability, Security</td>
</tr>
<tr>
<td>Wrong information</td>
<td>Design instead of requirement specified, Not what the customer wanted, Ambiguous requirements, Inconsistent requirements, Untestable requirements, Untraceable requirements</td>
</tr>
<tr>
<td>Extra information</td>
<td>Bells and whistles</td>
</tr>
</tbody>
</table>

**Missing information.** In the missing-information category, Digital Switch engineers list five errors they are working to prevent in future projects. The engineers came up with four methods they thought would be most effective in preventing their missing-information errors: checklists, standard formats, formal reviews, and prototyping.

Std 830 provides the checklist and the standard format (see Figure 1) that Digital Switch is seeking. Checklists and standard formats increase the probability of having some information in all requirements categories (like performance and interface).

Frequently, information in one of these categories will also be missing. For example, look at the error of missing quality requirements in Table 1. In that category, Digital Switch identified human factors, reliability, and security as important issues for future requirements specifications. To prevent these intracategory omissions, Digital Switch will use reviews.

The IEEE is developing a standard (P1028) to guide the review process. Until that standard is approved, Digital Switch is using its own review procedure. When a Digital Switch team reviews a requirements specification for a product, the builder, buyer/user, and verifier/tester of that product are expected to validate by signature that the requirements specification is complete.

For those products where Digital Switch and the buyer/user are not sure what the requirements should be, prototyping is being used. Now, Digital Switch is focusing on the prototyping of items like transaction screens and parts of databases that people interact with to derive more complete sets of interface requirements.

**Wrong information.** Digital Switch (as do most of us) has trouble keeping design information out of requirements specifications. (See the first error in the wrong-information category.) To prevent this error, Digital Switch has adopted a simple rule: Describe only data and functions that are visible outside the software to be created. That prohibits program and file structures and program interfaces — the kind of

---

**STANDARDS BULLETIN**

**Tools task force seeks help**

The Computer Society's Technical Activities Board has formed a task force to provide tool-related services to both hardware and software developers. Working groups will be established to pursue two startup activities: development of tool interface standards and development of a tools-inventory service. New participants are sought for these activities.

Three standards activities are getting under way now. The standards include:
- an interface between requirements definition tools and design tools,
- an interface between requirements definition and testing tools, and
- an engineering database scheme with which other tools may interface.

In the proposed inventory service, vendors would enter data about their tools into a database via dial-up telephone service. Inventory service users could dial up the system and query it for specific tool information by entering requirements into a fill-in-the-blank screen.

**X3 to work on Forte standard**

The X3 standards committee on information processing systems and the Computer Society are jointly working on an ANSI standard for Forte. The standard seeks to reconcile the Forte, Inc., implementation with the Forte Interest Group's version and will use the standards published by the Forte Standards Team as its basis. The effort is expected to be completed by Dec. 31.

If you are interested in joining the project, contact Elizabeth Rather, Forte Standards Team, Forte, Inc., 11 N. Sepulveda Blvd., Ste. 300, Manhattan Beach, CA 90266; (213) 372-8493.

**Standards update**

The IEEE Standards Board has approved three new standards and two new software-related projects. The standards are:
- P990, Recommended Practices on Ada as a Program Design Language,
- P1002, Taxonomy for Software Engineering Standards, and

The new software-related projects are:
- P1003.3, Standard for Test Methods for Measuring Conformance to Posix, and
- P1134, Standard for Personal Computer Basic Input/Output Software.

Copies of the new standards are available from IEEE Service Center, 445 Hoes Ln., Piscataway, NJ 08854-4150; (201) 981-1391. To volunteer for new projects, contact Helen Wood, ICST, National Bureau of Standards, B-154 Technology Bldg., Gaithersburg, MD 20899; (301) 975-3240; Compmail + h.wood.
internals that often slip into requirements. The rule also helps assure testability of requirements.

Including design information in a requirements specification is wrong because it restricts design trade-offs. If you tell a person what is required in a product as well as how to build it, you have specified requirements and designed the product. Design information is sometimes considered extra rather than wrong information, but however it is classified, it doesn’t belong with requirements.

“Not what the customer wanted!” Does that sound familiar? Digital Switch is addressing this correctness issue with two methods: reviews and prototyping. In addition, to reduce misinterpretations, requirements will be written in the customer’s vocabulary. This is hard on the requirements writer, but the effort usually pays dividends in customer satisfaction.

Digital Switch is working to prevent ambiguous, inconsistent, and untraceable requirements applying these rules:

1. Use data and function dictionaries that allow one — and only one — definition per term.
2. Use only simple sentences.
3. Use tables of information.

**FREE FACTS. FAST!**

We’ve just sped up the response time to our Reader Service cards, so now you can get information more quickly about advertisers’ products and services, about products we list in the New Products section, and about the software and books we review in our three reviews departments. Under our new system, the company will have your name and address just days after you send out the card. And we continue to pick up the postage!

While you’re indicating the products that interest you, please take a moment to circle the articles and departments that you liked in this issue. That will let us serve your interests better.

**We’d like to hear from you!**

---

**IEEE SOFTWARE**

---

**Figure 1.** Standards requirements specification format (from ANSI/IEEE Std 830-1984).

1. Introduction
2. General description
   2.1 Product overview
   2.2 User descriptions
3. Functional requirements
   3.1 Automatic functions
   3.2 Human-activated functions
   3.3 State-transition functions
   3.4 Error-response functions
4. Interface requirements
   4.1 Human interface requirements
   4.2 Hardware interface requirements
   4.3 Software interface requirements
   4.4 Communications interface requirements
5. Performance requirements
   5.1 Database, input, output loads
   5.2 Response times
   5.3 Resource usage limits
6. Constraints (environmental requirements)
   6.1 Location
   6.2 Hardware
   6.3 Software
7. Quality requirements
   7.1 Human factors
   7.2 Availability
   7.3 Reliability
   7.4 Maintainability
   7.5 Security

**If STRUCTURED ANALYSIS is your passion, you’ll love DeeDee™!**

THE DATA DICTIONARY management tool you’ve wished for!

- Syntax checking for data consistency
- Makes global changes
- Alphabetizes automatically
- Menu driven and configurable
- Tailored automated documentation
- Automated creation of sub-definitions
- VAX EDT-like editing of data structures

Keeps LARGE PROJECTS under budget!

**Powerful, Productive, Fast!**

DeeDee™... $2995
Ada and Pascal type declarations created with Mikt™ option... $995

**SCS inc.**  PO. Box 15367  FORT WAYNE, IN 46805 219/432-3975

---

Thanks to Pitch Thompson, Kermit Heaton, Rick Ross, Cary Barrineau, and Darrell Marsh at Digital Switch Corp. for their contributions to this piece.