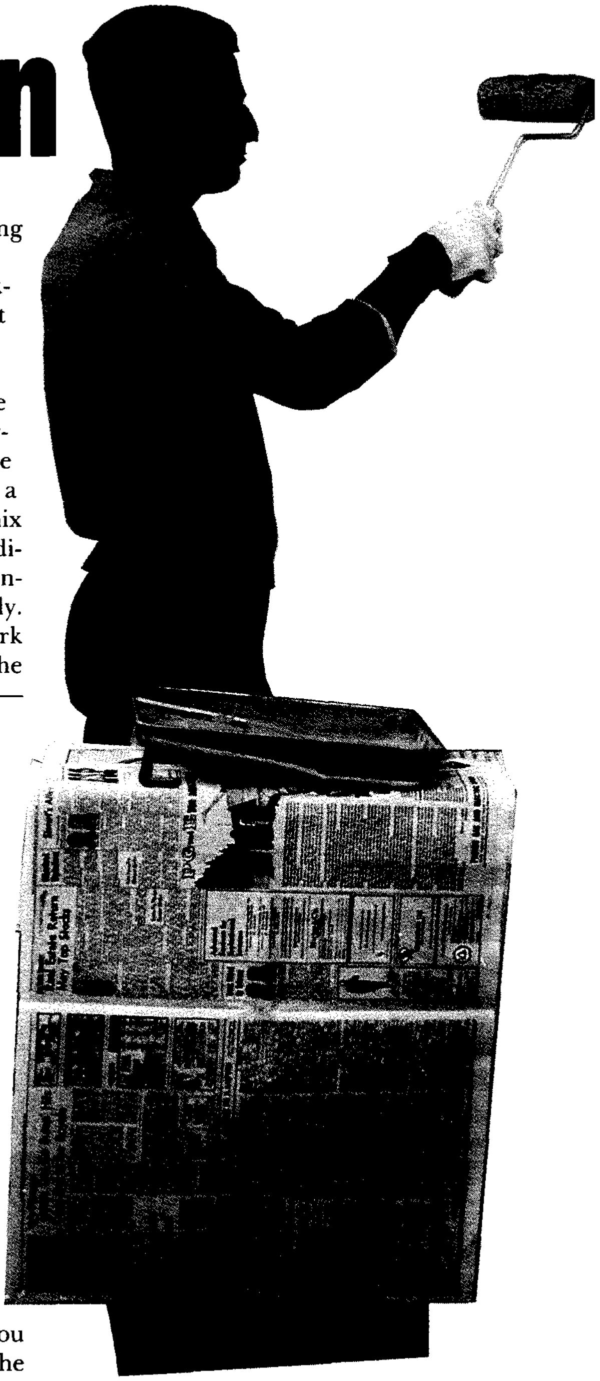
Nobody Reads Documentation



Finally, new and interesting ideas about documentation.

BY
Marc Rettig

It is kind of funny, really. Most documentation is written by technicians—not professional writers. And most technicians would include documentation among their top ten complaints regarding the software they use. Physician, heal thyself.

This column describes ideas and suggestions from current literature on software documentation. I hope they will change the way you think about documentation. If you are in the software field, it is almost certain that you will have to write documentation, for either your peers or your users. If you are designing software, you owe it to those you serve to gain an enlightened attitude toward documentation, recognizing the interconnectedness of the software, its documentation, and the help system. Otherwise, you are not a "practical programmer."

The State of the Art

Not long ago, most software documentation was simply a reference to all the commands in the system. Things have improved, thanks to an important lesson from the documentation community, which can be turned into a slogan worth remembering:

Manuals should not just describe the features of a system, they should help people get things done.

Manuals that follow this advice are called "task-oriented." As will be shown in the following, they have their own problems, but they are a big improvement over what used to be available. It would be a mistake to try to improve on task orientation without first understanding what is right about it.

It is easiest to explain a task-oriented manual by contrasting it with other styles. "Software-oriented" manuals discuss all the ways to "talk" to the software. The trouble is, it is necessary to understand the software to know where to look in the book. I once had a terrible experience with a Unix manual. I was using the vi text editor for the first time, and had managed to enter my text successfully. Then it was time to save my work and leave the editor. What was the command? The index was no help—it had only one entry for the entire section on editing. I was reduced to searching through the alphabetical command list until I came at last, red-faced and swearing, to the magic word: "zz."

"Menu-oriented" documentation has the same problem. It is common to see manuals that describe commands in the order they appear in the menu bar. But if what I want to do requires an option deep down in a nested menu, or worse, a series of such options, I will have difficulty getting help from menu-oriented documentation.

A task-oriented manual is more like a cookbook. It provides recipes for all the things you might want to accomplish with the software—showing how to use each command in context of the recipes. Such manuals are much more difficult to write, since you cannot rely on the structure of the interface to organize the documentation. You must go through an exercise in task analysis before you can start writing. But the payoff is worth the trouble: more effective software, happier users, and fewer questions.

PHOTO: PICTORIAL PARADE

The state of the art in commercial documentation seems to be producing the following:

- A tutorial
- A task-oriented user guide
- An alphabetical reference to commands
- A pictorial guide to windows, icons, and tool palettes
- A reference card
- Assorted specialized guides ("getting started," "installation," or vertical market applications of a general tool)
- An on-line help system, usually sort of hypertextish

This is a big improvement over what we used to get. I have hundreds of pages of slickly produced documentation on my shelf. Beautiful. But what do I use? The reference cards and the on-line help. And I am not alone. In fact, recent innovations in documentation are motivated by the following "obvious-now-that-you-mention-it" philosophy. . . .

No One Reads Manuals

Well, hardly anyone—consider some of John Brockman's important questions as presented in the SIGDOC '90 Conference Proceedings:

"A very disturbing fact of life that documenters have to face is that 'adults resist explicitly addressing themselves to new learning' (Carroll and Rossen, 1987). This doesn't mean that they burn books or begin cursing manuals as soon as they receive them, but they do perform a number of actions that can only be explained by this fact of adult mental life. For example, how does one explain the fact that when given even well-written and well-designed documentation, adult readers constantly skip ahead and begin to try to use the system without reading the whole manual? How do we explain the fact that adults are constantly guessing about what should and should not happen with a new system as soon as they begin learning? How do we make sense of the fact that 'learners at every level of experience try to avoid reading' (Carroll and Rossen)?"

Brockman, writing in a paper for SIGDOC entitled "The Why, Where, and How of Minimalism," elaborates with more quotes from an IBM researcher named John Carroll:

"It is surprising how poorly the elegant scheme of systems-style instructional design actually works. . . . Everything is laid out for the learner. All that needs to be done is to follow the steps, one, two, three. But, as it turns out, this is both too much and too little to ask of people. The problem is not that people cannot follow simple steps; it is that they do not. People are thrown into action; they can only understand through the effectiveness of their actions in the world. People are situated in a world more real to them than a series of steps, a world that provides rich context and convention for everything they do. People are always already trying things out, thinking things through, trying to relate what they already know to what is going on, recovering from errors. In a word, they are already too busy learning to make much use of the instruction."

Brockman summarizes the problem with five *very important facts* about adult learners. They

- Are impatient learners and want to get started quickly on something productive
- Skip around in manuals and on-line documents and rarely read them fully
- Make mistakes but learn most often from correcting such mistakes
- Are best motivated by self-initiated exploration
- Are discouraged, not empowered, by large manuals with each task decomposed into its subtask minutiae

These are some of the problems faced by documentation writers and software designers. How can you overcome these barriers, and help people learn to use your software well with a minimum of frustrations? We have task analysis to help answer the question, "What material should be included in the documentation?" The next question is, "How should the material be presented?" The minimalists think they have an answer.

Minimalism

Over the past few years, a group of researchers and writers have been promoting a writing style called "minimalism." Minimalism is motivated by studies of people's learning styles. I have already quoted John Carroll—one of the most vocal minimalists—as he describes the poor fit between adult learning styles and most documentation.

Carroll explains the minimalist solution to the problem this way: "The key idea in the minimalist approach is to present the smallest possible obstacle to learners' efforts, to accommodate—even exploit—the learning strategies that cause problems for learners using systematic instructional materials. The goal is to let the learners get more out of the training experience by providing less overt training structure."

So, a minimalist manual will have little or nothing in the way of overviews, summaries, introductions, indexes, and long windy chapters. It will have lots of material designed to encourage readers to explore the system on their own, to find the answer to puzzles, or to solve realistic problems with the software.

None of the minimalist literature contains specific guidelines for producing a minimalist manual. But John Brockman's SIGDOC article summarizes common features of minimalist manuals into these points:

- Cut secondary features of manuals and on-line documents—overviews, introductions, summaries, etc.
- Focus on what readers need to

The Best-Seller That No One Reads

Imagine a book produced by a team of professional authors. It was carefully designed and written to communicate effectively. Indexes were painstakingly composed and checked. The book was printed on high-quality paper, with two colors on almost every page. Millions of copies have been distributed—a huge best-seller for a computer book.

But few people have actually read the book! Throughout the world, copies wait on shelves for prospective readers to open them for the first time. Was all that effort wasted on the *Apple Macintosh System Software User's Guide*?

Apple worked hard on the documentation and interface for the Macintosh. Guidelines are published and enforced, and everything is tested by "typical users" before being released. Apple's writing staff is knowledgeable and experienced. The documents are attractive, have good reference aids, and are tailored to various audience levels.

Yet I so often hear people asking basic questions about the Macintosh that I began to keep a list. Here are some samples:

"If I use the print command, will it print the whole page or just what I see on the screen?"

"I asked it to print, and now I see a window that says 'LaserWriter 5.2!' What did I do wrong?"

"I put the document over the trash, but it won't go in! What's wrong?" (This person was putting the icon representing the document over the icon representing the trash can, but failed to make the cursor point at the trash. The effect was to make the two icons overlap—the document appeared to be sitting on the trash, refusing to be thrown away.)

"Why does it keep saying, 'The application is busy or missing?' I've opened this document before. . . . Can I recover my letter?" (The Mac associates documents with the application that created them. Asking it to open a document automatically starts the appropriate application. In this case, the required application was no longer in the disk drive. The user had been taught to open documents—not run applications—so the message was quite confusing. "What is an application? If it is busy, what is it doing? If it is missing, where did it go? I don't trust these computers.")

I know one documentation writer who painstakingly added a long chapter of "recipes" to a comprehensive user guide. Each recipe gave step-by-step instructions for using the software to accomplish some common task. Despite this, the company often gets calls and letters asking how to accomplish one of those very tasks.

Why is this? Well, I think the minimalists (see the column) are correct. The answer is something we often joke about, but something documentation writers have difficulty admitting to themselves: *most people don't read documentation.* They step up to a machine and start using it, turning to the books only when they are stumped or the computer does not conform to their expectations. Even then they first seek help from another person—the manuals are a last resort.

Short of making manuals that dispense dollar bills for every page read, I'm not sure it is possible to make a comprehensive manual that people will read. It is time to get creative. The availability of interactive tools gives us the opportunity to face facts and provide documentation that answers people's *real* questions:

"What do I do *now*?"
"What does *this* do?"
"How can I ______________ ?"
"What is possible?"

I solicit readers' votes for best and worst documentation. Tell me what works for you, and why, or what is useless to you, and why. Awards will be published in a future column.

know in order to immediately apply it to productive work.

- Test repeatedly during design; testing replaces any hard and fast rules and guidelines in the minimalist design philosophy.
- Make it easy for the reader of a page to coordinate the documentation with the screen information via pictures of screens or other graphics.
- Use what the readers already know by continuously linking new information to it (e.g., "given-new" metaphor approach).
- Encourage active exploration of a system via intentionally incomplete information.

Brockman cites four case studies in which existing documents were revised into minimalist documents, then tested against the originals for their effectiveness in training new users. The results are quite impressive. Following are some highlights from one study involving the tutorial for an IBM word processor.

Design Element:
"Slashed 75% of the pages by cutting previews, reviews, index, practice exercises, troubleshooting, 'welcome to this application' introductions, and most pictures of screens."
Rationale:
"The rule of thumb in minimalist design is to try first to cut and condense text and other passive components, but the goal of this is to enrich the training experience. The trick is to give the learner more to think about, but less to overcome."

Most developers treat documentation as a separate product, a part of the package that they deliver to their customers along with the software.

Design Element:
"Left procedural details deliberately incomplete to encourage learner exploration, but gave 'enabling hints.'"
Rationale:
". . . readers who learned procedures by working exercises that forced them to independently apply the information in the manual performed significantly better."

Design Element:
"Used open-ended exercises called 'On Your Own.'"
Rationale:
"Procedural knowledge is difficult or impossible to write down and difficult to teach. It is best taught by demonstration and best learned through practice."

Design Element:
"Kept chapters brief (averaging less than three pages)."
Rationale:
"The units must be very streamlined so that learners are not likely to skip around within them; the organization of the units must be very simple so that learners can more successfully skip over units or skip among units."

The preceding gives just a taste of the minimalist approach. The rest of the article makes very interesting reading, as does Carroll's book. Which is fine, but does it work?

Brockman reports that tests between the original and revised manuals found that using the minimal manual resulted in:

40% less learning time
58% more tasks completed
93% more tasks completed per unit of time
20% fewer errors and 10% less time recovering from errors
80% more "exploratory episodes"

I do not know the testing methodology or the numbers behind the statistics, but these are impressive claims. The claims are supported by three other case studies in the same article, each at different companies. I suspect the minimalists are on to something.

But they do not have a monopoly on good advice for designers and documentation writers. Even people outside the computer industry have useful ideas about such things. . . .

Document as Product, or Vice Versa?

Most developers treat documentation as a separate product, a part of the package that they deliver to their customers along with the software. Try changing your way of thinking—see if you can design a software product that is its own documentation.

In the SIGDOC '90 Proceedings, Lorraine Wilkin and Wendie Wulff describe an interesting case study that illustrates this idea. It seems a company called Mallinckrodt produces a kit for cleaning up spills in chemical laboratories. The kit includes things to protect the user, to absorb and neutralize the spill, and to clean up the mess. Taking the "documentation as product" approach, the kit also included a 30-page manual, organized by type of spill. So if you knocked something over in the lab, you would select the appropriate kit, find the manual, and search for the instructions for cleaning up whatever you spilled. Of course, lab workers are no different from computer users when it comes to documentation—they don't read it until they absolutely have to. In some cases, this caused enough delay to increase the hazard. Worse, when the custodian knocked over the beaker filled with acid, there was no way for him to tell which kit to open, and the instructions were difficult for the uninitiated to understand.

Mallinckrodt changed their way of looking at things. They decided the product should be its own documentation, and created packaging which made the booklet unnecessary. The new kit comes in a box with huge lettering that identifies what it is for: "ACID SPILL." Inside the lid are clearly captioned pictures showing what to do. If you are in too much of a hurry even to look at the pictures, you still won't go too wrong. Inside a box labelled with a giant "1" is a pair of rubber gloves. Pretty clear what to do. Then there are bottles labelled "2" and "3." Spread them on the spill. Finally, a whisk broom and dust pan labelled "4." Get it? The product is its own documentation.

So, here is a suggestion for software developers. Design your software. Write instructions for using the software to accomplish some important task. Now ask, "how can I embed these instructions in the software?" There are plenty of examples. Some tax programs present you with an image of a form—you already know how to fill in a form, and it doesn't take long to notice that some boxes are "magically" filled in for you. Some data management programs help you define a database by giving you a series of forms to fill out. You are not aware of it, but by filling in the forms you are defining a database schema. In the same spirit, many graphical interfaces try hard to make every

Try changing your way of thinking—see if you can design a software product that is its own documentation.

command and function somehow visible—as a button, a menu command, or as an item in a tool palette. The only active items are the ones that are appropriate in the current context.

Imagine an interface that acts like the Mallinckrodt packaging. You would choose a task from a list. "Look up account by social security number." A big red "1" would appear by the "Find" button. Click it, and a "2" appears by the social security field in the entry form, and a "3" beside the return key in a little picture of the keyboard. Who needs a manual?

Documentation Cannot Make Up for Bad Design

You already know this—I do not expect argumentative letters over this point. But sometimes it is difficult to put the notion into practice. In many companies, the system designers are located at a distance from the documentation staff. Rather than including a documentation specialist on the development team, companies separate the two functions, making it easier for designers to think, "we'll explain it in the documentation."

This is the number-one point in Apple's *Design Principles for Online Help.* "Before setting out to build a help system that 'explains' a difficult interface, try to identify what makes the interface difficult, and fix the problems. Once you have made your interface as clear as it can be, develop a help system that aids users as they work."

It is easy to form a habit of leaving explanations to the documentation. Too often your users will encounter the confusing part of the system before they read about it in the manual. They will be left holding a sloppy design and nursing a bad attitude toward the software. They will be far happier with a system that makes its functions and use clear by its appearance. (See Donald Norman's book, recommended at the end of the column.)

More Fun Than You Think

Most software professionals think of documentation work as drudgery. So did I, until I began to treat it as part of the system design—part of the user interface. I am quite excited by the possibilities of the ideas of "product as documentation," minimalism, and others described in the resources below. Both Brockman and Edmond Weiss emphasize that documentation is an engineering process with many parallels to software development. It should be designed, it should accommodate the needs of its audience, it should be tested as thoroughly as the software, and it must be maintained. My advice to programmers: make it a point to acquaint yourself with what the documentation specialists have to say. It will improve the quality of your software.

Resources for Documentation Designers

Brockman, R.J. The why, where, and how of minimalism. In ACM SIGDOC '90 Proceedings (published as SIGDOC Asterisk 14:4), pp. 111–119.

Brockman, R.J. *Writing Better Computer Documentation for Users: From Paper to Online.* Wiley-Interscience, 1986.
A nice complement to Weiss. Similar approach, but more extensive.

Carroll, J.M. *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill.* MIT Press, Cambridge, Mass., 1990.
The primary source on minimalist documentation design.

Carroll, J.M., et al. The Minimal Manual. *Human Computer Interaction* 3, (1987–88) (1987–1988): 123–153.
This is the source of the case studies cited in Brockman's SIGDOC article on minimalism.

Desberg, P. and Taylor, J. *Essentials of Task Analysis.* University Press of America, Lanham, Md., 1986.
A little blue book that teaches you how to break big tasks into lots of little tasks. And it follows its own advice—it is basically a task analysis of task analysis!

Weiss, E. *How to Write a Usable User Manual.* ISI Press, Philadelphia, Pa., 1985.
Good advice and "how to" information for incorporating documentation into the software development process. Useful forms, guidelines, and procedures. Takes a task-oriented approach.

Wilkin, L. and Wulff, W. Document means more than manual. In *ACM SIGDOC '90 Proceedings* (published as *SIGDOC Asterisk 14,* 4), pp. 79–86.
This is the source of the "product as documentation" idea discussed in the column.

Other Useful Goodies

Norman, D. *The Psychology of Everyday Things.* Basic Books, N.Y., 1988. Now available as *The Design of Everyday Things.* This book shows us how simple visual cues can make something's operation perfectly

clear, by drawing on the user's previous experience of the world. That is, the appearance of a good tool clearly shows its proper use, making function plain through good visual design. While only a small part of the book actually discusses computer software, the entire book speaks very clearly about what software designers should be doing.

Heckel, P. *Elements of Friendly Software Design.* Sybex, San Francisco, Calif., 1991.

While not explicitly about documentation, this book does an unequaled job of explaining how to think of software design as an exercise in communications—exactly the same idea that motivates the other sources described here. The 1984 first edition (from Warner books) was widely acclaimed. This is a must-read for software developers, and I recommend it every chance I get.

Finally, three books about writing English—the implementation language of documentation. Just as you can learn to write good C by practicing what you read in a book, you can learn to write decent English the same way.

Elbow, P. *Writing Without Teachers.* Oxford University Press, 1973.

Great for encouragement, motivation, and advice. Great exercises for overcoming writer's block.

Strunk Jr., W. and White, E.B. *Elements of Style.* MacMillan, N.Y., 1979.

Less than 100 pages, and worth reading at least once a year.

Zinsser, W. *On Writing Well.* Harper and Row, N.Y., 1985.

A great book about writing.

Loose Ends: Team Management and Software Testing

If you were interested in the ideas described in my column on "Structured Open Teams" (Oct. 1990), you should know about the People and Systems Conference to be held in Boston, Sept. 10–13, 1991. The entire conference focuses on management of software development. Larry Constantine will be there teaching about Structured Open Teams, and many others will share their experiences with team management and innovative management techniques. Contact Software Development Conferences in care of Miller Freeman Publications at (415) 397-1881.

My May 1991 column, "Testing Made Palatable," generated a lot of inquiries about our Test Manager for Smalltalk. The source code and documentation for the Test Manager is now available to all takers. Email or surface mail me for information. **C**

The ***Practical Programmer*** *wants to hear your stories. What worked for you, and why? What didn't work, and what were the horrible results? Forthcoming columns will discuss visual design, metrics, and the development process. Send your braggardly tales and autopsy reports to:*

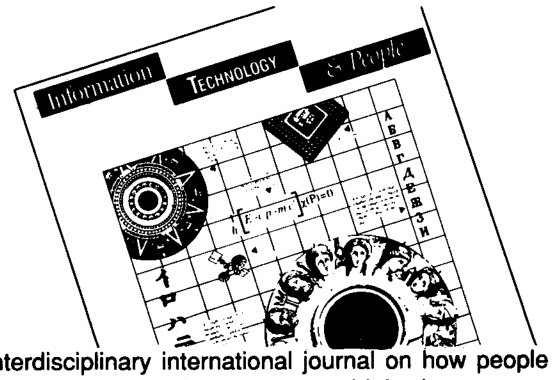
Marc Rettig
Academic Computing
Summer Institute of Linguistics
7500 West Camp Wisdom Road
Dallas, TX 75236
Internet: marc@txsil.org
Compuserve: 76703,1037
The Well: mrettig

Marc Rettig *is a member of the technical staff at the Summer Institute of Linguistics, and a freelance explainer of things. He serves as Technical Editor of* Database Programming and Design *magazine.*

USERS?!

What do you know about users? *How do you know what you know? Do you have a workable set of distinctions? Do you relate to users as real people? Or are you so confused you want nothing to do with them? Can you deal with users only through a one-way mirror, or plotted on a flow-chart?* Yikes! You need...

Information TECHNOLOGY & *People.*



Truly interdisciplinary international journal on how people use information technology: ways to think about users, ways to talk/listen to them, ways to design for/with them. The journal is peer reviewed. It draws from computer science, academic MIS, and social science to examine information technology effects and design, with emphasis on the organization and group.

$130 annual subscription, published quarterly
Box 500, 19363 Willamette Drive
West Linn OR 97068 phone/fax (503) 656-7108

Circle #52 on Reader Service Card