VOLUME 19
NUMBER 1
March 1995

THE JOURNAL OF COMPUTER DOCUMENTATION

Association for Computing Machinery
1515 Broadway, New York, NY 10036

SIGDOC Special Interest Group for Documentation

Lee, R.H. (1994). Design issues for hypertext coaching systems. Paper presented at the 42nd Technical Writers' Institute, Troy, NY: Rensselaer Polytechnic Institute, June 7-10.

PC Magazine. (1988). "PC Magazine Survey." March 29, pp. 40-41.

Redish, J.C. (1993). Understanding readers. In C. Barnum and S. Carliner (eds.), *Techniques for Technical Communicators*, pp. 14-41. NY: Macmillan.

Redish, J.C. (1988). Reading to learn to do. *Technical Writing Teacher*, xv(3), Fall, pp. 223-233; reprinted in *IEEE Transactions in Professional Communication*, 32(4), December 1989, pp. 289-293.

Seeking the Future of Software Documentation

by David K. Farkas
Department of Technical Communication
University of Washington

Edmund Weiss is an iconoclast; he is more than willing to challenge established points of view. His article is complex and includes some debatable claims and, perhaps, some idiosyncratic viewpoints, but it's also shrewd and provocative. Weiss is one of the Grand Old Men of the profession; he reads broadly, thinks deeply, and has watched the computer industry and especially the documentation side of it unfold decade by decade. Ed Weiss is definitely someone worth hearing from.

Weiss presents an entire history of computer documentation and its ties to user interface design. His history takes us from the crude beginnings of documentation and interface design in the 1960s and 70s, to Weiss' Golden Age—the 1980s—and into a period of decline and uncertainty.

He speaks disparagingly of the present-day "balkanization" (that is, fragmentation) of documentation into such disparate elements as balloons, coaches, and wizards (in addition to standard help and print documentation). He envisions a future of disempowered writers who do not exercise their craft as much as they feed information into monolithic hypertext databases that users are left to navigate through in search of relevant information. He offers some hope but not great assurance that the new adventurous, aggressive computer users of the future will be equipped to deal with this situation.

In my commentary I discuss what I take to be Weiss' most significant claims. I then offer a different and more optimistic answer to the question of where we are now and where we are headed. I uphold the mainstream viewpoint in which the many forms of documentation are seen, not as inchoate fragments, but as reasonably effective components of a coordinated documentation set, components that act in concert to provide different kinds and levels of support. If this viewpoint is well founded, documentors have not become disempowered yet.

Complexity of Software Products

Few would argue with Weiss' contention that the software applications we find everywhere in offices and very often in homes have grown extremely complex and, indeed, for many users too complex. Word processors, spreadsheets, and other standard applications have very powerful features, offer numerous ways to perform a single task,

and very frequently allow users to customize the interface in a variety of ways. For many users life would be simpler if they worked with less complex products. Furthermore, the documentation for these highly complex products is necessarily massive (very thick manuals, large help systems, or both), and this makes life more difficult both for documentors and also for the unwilling readers of these massive documentation sets.

Weiss blames software developers, who choose to compete on the basis of ever more bells and whistles, as well as trade magazines, which publish feature-by-feature comparisons and suggest that more power is always better. But most of all he blames computer users, ourselves. Repeatedly, "lite" versions of software products have fizzled in the marketplace, and Weiss sees no sign that computer users are about to become less enamored of extremely complex software.

There is a large element of truth in Weiss' argument. On the other hand, there are certainly successful, favorably reviewed products whose hallmark is a limited feature set. Also, contemporary interfaces allow users to decide how much complexity they want to tackle at any time. Despite the power and complexity of Microsoft Word, a novice can write simple documents using nothing more than the same basic commands found in the stripped-down word processor Microsoft Write (or even Notepad). Looking back a dozen or so years, my first word processor, AppleWriter II, had very limited capabilities but demanded significant learning just for writing basic documents. Weiss counts 48 ways for the user to select a single word using Microsoft Word, but of course most users get along fine using just a small subset. Along the same lines, novices do not need to customize their interface. The benefits of customization, however, are considerable: if a particular user is going to make extensive use of equations, it is nice to have the equation features readily accessible.

Re-assessing GUI Interfaces

Weiss expresses serious reservations about the usability of today's graphical user interfaces. These reservations are best appreciated in the context of his historical narrative.

The command-line interfaces of the 1960s and 1970s were empty and unstructured and required users to learn and remember numerous commands. People hated manuals but had no thought of using a computer system without the documentation. In the 1980s hierarchical menu structures vastly improved matters. Users could choose from a menu of choices rather than memorize commands. These hierarchical, character-based interfaces controlled and protected the user in a paternalistic manner.

These interfaces significantly reduced the need for documentation. Furthermore, these interfaces made possible very accessible, brief, and effective field-based online help. Manual writers, also following a paternalistic model and adopting sound design principles, became good at guiding users through task-oriented procedures with a minimum of distraction and confusion.

To Weiss, GUI interfaces are inherently unstructured and, in a sense, return us to the bad old days of the 1960s and 70s. GUI interfaces (in conjunction with the complexity of software) make it almost impossible to protect users from themselves. Task-oriented documentation becomes difficult or impossible to write. Weiss asserts, "the documentor is forced to describe objects and process molecules and hope users will make sensible strings of them. Rather like a DOS reference manual!" Weiss sees the many new kinds of documentation (balloons, wizards, etc.) as a consequence of the largely dysfunctional nature of GUI interfaces.

I think it is useful for Weiss to remind us that GUI interfaces regularly confuse users and do not solve all of our usability problems. But Weiss may be overstating the usability of the older character-based, non-windowing applications that he ran years ago. Furthermore, Weiss' distinction be-

tween the clean, hierarchical applications of the past and the new GUI-based applications should be moderated. GUI applications, built as they are around menus, commands, and dialog box choices, are themselves hierarchical—the difference lies largely in their use of visual elements (buttons, check boxes, etc.) rather than successions of menu choices. Later on I discuss further Weiss' view of how GUI interfaces have affected documentation.

An Emerging Generation of Users

Weiss sees the emergence of a new kind of computer user who was raised on computers, learns independently, and regards computer use not simply as a way to get work done but as an adventure. These are the people who always customize their screens. They are happy with unstructured GUI interfaces. They do not need paternalistic documentation; in fact, they are even more resistant to reading than their predecessors.

Pushed far enough, this claim would be cause for optimism. The problems of very complex products, puzzling interfaces, and massive, hard-to-search help systems and manuals would all give way before the prowess of the emerging super-user. Weiss, however, does not go nearly this far; he really doesn't tell us we can count on a world populated by these super-users. I suspect he believes that these users will remain only one part of a more diverse user profile, a profile that includes many less capable and less adventurous users. It may well be, however, that super-users are becoming a larger portion of the total mix.

Contemporary and Future Documentation

Clearly, Weiss is not happy about the state of documentation. Most of his reasons I have noted. Because of the size of the documentation set (a consequence of complex products and GUI interfaces), readers have problems finding their way through documentation and hate reading it. He believes that GUI interfaces, because they are unstructured, because they allow users to do one thing many ways, and because they use many small graphical objects, have made task-oriented documentation difficult or even impossible to write and have led to a proliferation of disparate forms of help.

Another major objection is that users of help systems now can alter the help window size and otherwise customize the appearance and behavior of their help systems. (He will be still less happy with Windows 95 help, which allows users to adjust the font size and invites them to change colors.) For Weiss, to customize the help interface is to throw off the writer's control and limit the degree to which writers can exercise their craft to guide and protect users. Weiss fears user-customization even in print documents. He foresees users, empowered by technologies like SGML, overriding writers' decisions about fonts and layout. One response to Weiss' argument is that users need the ability to control the size of their help windows and, to a lesser extent, colors—if only because computer displays vary in their dimensions, resolution, support for color, and so forth.

Weiss sees us at the verge of an era of "virtual documentation," in which users create their own reading experiences as they search through databases in which innumerable small molecules of information are shaped by writers only in a minor way. This database may well include standard help, balloons, wizards, and all the other components of a documentation set, but since documentation in general has devolved into innumerable small, uncrafted molecules, the sum effect from the user's point of view is one virtual document that is monolithic in character. Ultimately, much of this virtual information might be generated automatically—without any involvement from the documentor—directly from the application's code.

Here, Weiss' notion of the new user comes into play. In part, he points to a reasonable match between the new super-user and the

largely "authorless" documentation. But he doesn't express much confidence for the future, very possibly because he is not confident that all users will be so adept with computers.

An Alternative Vision of the Future

Weiss' historical narrative, assessment of the present, and cloud-filled vision for the future are well worth contemplating. I'd like to present a different perspective, however. My starting point is Weiss' complaint that documentation—and help in particular—now consists of many fragmented components (Weiss' Balkan nations), whose distinct roles are largely lost on the user. In contrast, I adhere to the idea of the coordinated multi-component documentation set. To support this position, I review some of the major forms of contemporary documentation, acknowledging their various weaknesses but asserting that collectively the contemporary documentation set offers real support for different classes of users and different kinds of user problems. As part of this review, I suggest how the future of documentation may be shaped by advancing technology.

Standard procedural documentation

Standard procedural documentation is typified by the extremely prevalent Windows 3.1 help. The print user's guide is another major form. Large help systems come closest, I think, to Weiss' vision of monolithic hypertext databases. With these help systems, users encounter hundreds or thousands of lengthy, text-laden topics. In large part because of re-sizable help windows, the layout of these topics is rudimentary in comparison with high-quality print manuals; writers, therefore, cannot guide and assist users through refined formatting. Because of the complexity of these help systems, the topics are very richly linked, creating navigation problems. Large help systems may indeed seem like trackless, featureless information databases to many users. On the other hand, contents topics, search keywords, pop-ups and jumps are merely electronic analogs of tables of contents, indexes, footnotes, and cross references. Also, many help systems have browse-sequence buttons or some other means of letting the user navigate through a succession of help topics according to a path prescribed by the documentor (the equivalent of turning pages).

Procedure topics, I think, are usually task-oriented and provide a lot of usable information for those who will seek it out and read it. The topic title, the paragraph or two of text that follows it (sometimes called the conceptual element), and the subheadings (usually in infinitive form) that introduce individual procedures in a topic all explain the purpose and essential concepts. The steps provide explicit how-to information.

One major drawback of complex products and GUI interfaces is the considerable quantity of what can be called "special case" information. This includes multiple ways of doing something, different starting places, variations on the main goal of the procedure, and special conflicts or hidden problems that users need to be informed of or warned against. Special-case information typically appears in various kinds of notes, but also in steps or in the conceptual element.

Special-case information is one of the key dilemmas in the documentation field. How much do we provide and how do we provide it? Users find special-case information burdensome—until the moment when they need one of those items of information. The writer's presumed solution, that users will simply scan a lengthy procedure topic for the information they want, does not seem to work to the satisfaction of many users. The impatience of these users bolsters the argument for minimalist documentation that supports only mainstream situations.

Step-by-step guidance via a succession of prompts

Weiss does not acknowledge the role of the various documentation types that have in

common the careful guidance of users, the paternalistic control that he feels we are losing. Whereas in standard help each procedure topic covers a full procedure, step-by-step prompts direct the user to perform a single action or just a few simple, closely related actions. Often, when one task has been completed, the system prompts the user to choose from a set of related tasks.

One venerable form of prompting is the online or print tutorial; tutorials, however, carry a major limitation in that users typically must work on "canned" tasks rather do their own work. Today, however, users can enjoy the benefits of step-by-step guidance while they do real work. The coachmarks in Apple Guide actually mark the place on the interface where the user needs to click next. Wizards replace normal interface operations with a simpler set of choices, each fully explained. There is, of course, a trade-off here; these forms of step-by-step guidance mandate certain patterns of work and can restrict the end-products the user is able to create.

Annotations on the interface

Another form of help are the annotations that appear directly on the interface at the place where the user rests the pointer. Examples are Apple's balloon help and Lotus' bubble help. Usually the help information consists not of prompts or steps but of brief descriptions of the function of an interface element. Balloon-type annotations were initially regarded as aids to novices; in fact, they are at least as well suited to confident, independent computer users who dislike standard help but occasionally would like very convenient and brief explanations of some check box, option button, or other interface element. Because they (typically) offer only functional descriptions, balloons call for inferring and other kinds of problem solving. They are, in fact, a kind of minimalist instruction.

One of Weiss' objections to GUI interfaces is their reliance on cryptic icons (symbols). At the same time, Weiss disapproves of the new forms of documentation that have arisen in the era of GUIs. But Weiss should welcome annotations; they are a simple, elegant solution not only to the problem of cryptic symbols but any form of exceedingly brief interface text. Balloons-type annotations are, in fact, fairly similar to the convenient field-level help that Weiss fondly remembers in the pre-GUI era.

Adaptive or intelligent help

This last category is the fuzziest since its defining trait is not a particular design but particular computer technologies, some in existence, some emerging, others merely projected. The basis of this form of help is integration (and communication), whether at a modest or a very high level, between the application and the help system. In other words, in varying degrees the help system recognizes what the user is doing in the application and responds appropriately.

Right now we see a rudimentary form of this integration operating in the other three models. In standard help, the user can open a dialog box, press the F1 key (or a help button), and get a help topic keyed to that dialog box. Apple's balloon help knows when a command is unavailable (and grayed out) and will explain why it is unavailable. Various step-by-step prompting systems, such as Cue Cards, can (sometimes) detect when a user has failed to properly carry out a step and offer a corrective prompt.

With even a moderate advancement in integration (and a lot of work), we might alleviate the vexatious problem of having to document so many special cases. In other words, a certain warning would only appear if the user were subject to that danger; a certain variant goal would only be mentioned if the goal made some sense in the user's current context.

With breakthrough advances in the relevant technologies, we might see a truly new model. We can envision help that carefully monitors a long succession of user actions, builds a profile of the user's computer behavior, infers the user's goals and computer

ability, and offers pertinent advice. Unlike the very limited intelligence of, say, a Cue Card, this form of help would identify and address subtle user errors and many forms of user uncertainty. Here we approach the dream of documentation that functions as a cooperative, expert assistant who is always available to the user.

In this scenario, standard help, balloon-type annotations, wizards, and other forms of documentation might indeed be supplanted by this one monolithic but extremely powerful and effective form of help. Here the role of documentors would not be to write documentation but rather to work with programmers to plan out how this expert assistant will interact with users under varying circumstances.

Short of this technology, software products will continue to offer various forms of distinctive help, each with a certain primary strength: for standard help it is the completeness of its user support; for balloon-type annotations, it is the rapid access to information and support for problem-solving; for step-by-step prompts, it is the protection and guidance of the user. Each still requires all the authoring skill that we can apply.

Not Post-Usability, Just Different Usability

by R. John Brockmann
University of Delaware

To ensure that their critical messages are heard by an audience that is often is predisposed to disagree, prophets have shouted, performed absurd acts, and invented shocking words and phrases. Edmond Weiss wants to join such a prophetic rhetorical tradition in our field of computer documentation and, specifically, in the area of usability. He shouts in his article using pejorative words such as

- stubborn,
- paternalistic, and
- menu-trees (a no-no we've all learned to abhor over the last decade [1])

to label his opponents, 80s style usability adherents. His shocking words and phrases draw dramatic, if somewhat fallacious, either-or distinctions such as:

- Blank screen interface, good; menus, predefined data entry screens, intelligible screen prompts, bad;
- user-friendly, bad; feature complexity, good;
- 80s usability directed to novices, bad; 50s, 60s, 70s usability, good; etc.

Beneath all this prophetic bluff and bluster, however, Professor Weiss is accurate in describing how the environment of usability and the genre of user manuals have transformed in the last decade

- from a time of technology introduction to a time of technology acclimatization;
- from a time of technology one-person-does-it-all (owner-operator-mechanic) to a time of technology nested within a growing support infrastructure (owners, operators, and mechanics distributed to different roles and specialists); and
- from a time of author-reader ultimatum to a time of author-reader negotiation.

Thus Weiss calls for a new sense of usability, authorship, and text. Not only is his call