

edmond weiss consulting

115 Cooper Rd., Voorhees, NJ 08043
856-753-3440/edweiss@aol.com/www.edmondweiss.com

The Retreat from Usability:
User Documentation in the Post-Usability Era
The Journal of Computer Documentation, (ACM
SIGDOC), March 1995*, pp. 3-18

Edmond H. Weiss, Ph.D.

* Although originally published in 1994, this article accurately predicts the direction of software development. Office 2007 is vastly less usable than Office 97; indeed long-time users of WORD need a manual (none provided) to perform simple operations in the latest version.

The Retreat from Usability: User Documentation in the Post-Usability Era

Edmond H. Weiss, Ph.D.

Associate Professor of Communications,
Fordham University Graduate School of Business Administration

Abstract

During the 1980s, developers and documentors collaborated on a joint mission: to make applications (and their manuals) as usable as possible: easy-to-learn, easy-to-operate, and therefore more useful. In recent years, however, developers have substantially retreated from this protective approach to users, placing a greater emphasis on flexibility, feature-richness, and customizability, none of which is consistent with the traditional, technical communicator's model of usability. New conceptions of software, and new expectations about users, may result in a "post-usability era," in which documentors will probably need to moderate their traditional rhetorical and pedagogical roles as protectors of users and advocates for the primacy of ease-of-use.

The Usability Paradox

Usability is a paradox. When we think of it as "friendliness" or "ease" of use, we forget that, at least in the 1980s, we made systems more usable by restricting the choices of users, by replacing their unencumbered blank screens and command lines with structured and unyielding menus and predefined data entry fields and intelligible screen prompts.

Cheerful language should not obscure the obvious facts. What we have traditionally called a *friendly* interface (at least until about 1990) is really a *stubborn* interface: a short, inflexible menu of choices, often leading to another and another. From this perspective, menus do not give choices; they limit them. Pull-down menus do not just give the few available options; they "gray-out" the options that are invalid or inappropriate. That is, to the extent that unencumbered users are free to make serious blunders or paint themselves into inescapable corners (as with the libertarian C-prompt), to that extent usable design usually has entailed the restric-

tion of the users, for their protection and to enhance their comfort and productivity.

During this period, roughly from 1980 to the present, *usability* and *usability testing* have been closely connected. The methods used in the latter tend to operationally define the former. For human factors specialists and for interface designers, the testable domain of usability has been somewhat broader than for technical communicators. Booth, for example, and more recently Rubin, [1] include in the provenance of the term not only ease of operation and ease of learning, but also general product usefulness and even attractiveness (likability). In contrast, when technical communicators use the term they tend to restrict it. Leonard and Waller [2] remark that "a simple definition of a usable software product is an application that is easy to learn and easy to use." Similarly Rubens and Rubens [3] propose that "usability testing helps determine whether or not the product you develop will be easy to use for a specific audience."

This difference in perspectives is easy to understand. Historically, a central charge of technical communicators has been to ameliorate and simplify in-place software and systems. In many organizations they have served as the users' advocates, often the only ones willing to defend ease of learning over other design objectives. But this emphasis, understandable in the context described below, excessively restricts the vision of the technical communicator. Put simply, the notion that easier is always better, or that users do not want hard to use applications, may put technical communicators out of touch with the current and coming eras of software development.

Ending the Dependence on Manuals

The 1980s pursuit of usability was largely an assault on assumptions in the computer technology of earlier decades. In the 60s and 70s, technology, especially software, was presumed to be inherently obscure and difficult; it could be used only after extensive training and only with considerable memorization. Because there was nothing intuitive or easily understood, because the interface was generally empty and unstructured, the burden of training the user and providing auxiliary memory fell typically on the manuals. The manual, like a prosthetic that shapes the user's thinking into a profile that matches the profile of the technology, had to be present at all times. Users and operators, even the best and most experienced, were manual-dependent; their books were peripheral devices; almost no one could work without them.

The 80s paradigm was different: the "intuitive interface." In the PC epoch, writers and psychologists attacked the notion that software is inherently difficult. Cynical human factors critics sometimes complained that the users' manual was a compendium of everything wrong with the user interface. The notion flourished that, with the right screens and pointing devices, software (and other programmable technology) could be made "intuitive." (By "intuitive" people in the industry really mean "transparent"; un-

fortunately, though, the latter term is unavailable because of its chronic misuse by programmers as a synonym for "invisible.")

The overarching ambition was that, by replacing remembering with recognizing (through various fixed and task-linked menus), by replacing the typing of long strings with short strings (menu letters, F-keys) or with point-and-click (cursor bars, arrows), and by otherwise reducing the causes of confusion and impasse, we could produce computer systems and devices that could be used with little or no orientation and only infrequent recourse to manuals.

Nearly everyone involved in the development of technology—programmers, documentors, marketers—was joined in pursuit of a common mission: to make computers and applications no more difficult to use than household appliances, if not easier. Developers often used the analogy of rental cars, reminding us that almost anyone could operate any model of rental car without instructions and almost without any trial and error.

Important to this plan was breaking the grip of users' manuals, almost universally despised, and ending the need to read books as a precondition to learning an application. There was an evangelical mood to this mission, a sense of common commitment to a noble goal that would make people ever more happy and productive. By the end of the decade, even educated people—not just those who had difficulty with technical publications—sometimes boasted about their reluctance to use manuals. The idea gained acceptance in some circles that products which needed manuals or force us to consult them are "underdesigned" products.

This new, usable approach to software engendered a new relationship between system developers and "information developers"—the new name for documentors and technical writers. Instead of writers becoming involved at the end of the development cycle, where their task was to produce "comprehensive" descriptions of the new

product, they were often now on the development team, participating in the design of the interface.

Because preparing manuals and instructional materials gives writers special insights into what makes the system difficult or error-prone, the "information developer," acting as advocate for the user, especially the beginning user, was to press for ease-of-use in the product, often even writing the prompts, menus, and messages that constituted much of the user interface. (As a fortunate coincidence, the declining price of computer resources made it feasible to replace most coded and indecipherable messages with natural language sentences.)

Having exerted their influence on the design, writers would then "document" at the margins: teaching people how to use the generally intelligible interface and elaborating only on especially difficult or error-prone aspects of the system. Further, they would lobby to have these weak spots corrected in the next version, so that the need for the explanations would be gone.

Occasionally there was some stress between the developers and writers. Overzealous documentors seemed preoccupied with the needs of the brand-new, inexperienced, even hopelessly inept user. And programmers, in contrast, tended to think of users as somewhat more resourceful and willing to learn an arcane trick or two. (This conflict, as it turns out, is at the heart of the retreat from usability.)

In some cases, there was also a conflict when writers wanted to take control of screen messages, especially help screens, while developers wanted to restrict writers to the domain of manuals and training materials—to paper. For the most part, though, the collaboration was successful and the result was a kind of system that gave plain instructions, asked intelligible questions, provided information when needed, and generally freed the user from the need to refer to manuals.

This sort of system or application was nearly always a character-based program, running either on a nongraphical PC or even on a relatively dumb terminal. Its two central "usability" elements, the features that made it workable even without reference to a manual, were its use of menu trees (linked chains of simple choices) and context-sensitive help (popup screens containing either a list of allowable entries for a given field or an instruction for deciding what to type in that field). This technology was able to automate and simplify nearly every business and administrative function, from order entry, to flight reservations, to banking transactions, to point-of-sale commerce. The operators of these systems tended to be clerks and low-level administrators and, after a brief orientation course, they could expect to work for long stretches without consulting a publication. (Indeed, problems were almost always the result of system malfunctions, rather than the operators' mistakes.)

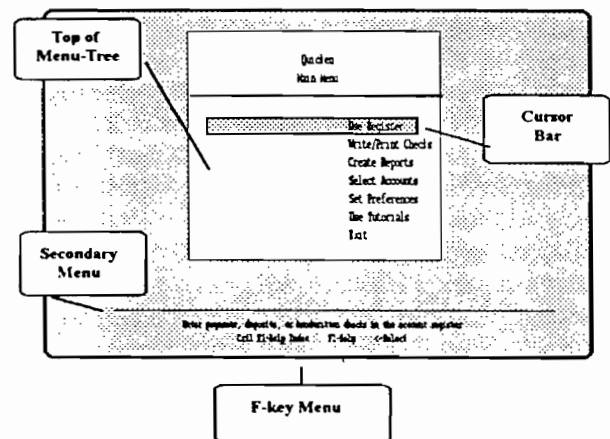


Figure 1. Representation of Quicken for DOS (Menu)

Consider, for example, this rendering of the opening screen of the DOS version of Quicken (Figure 1). In its simple, uncluttered presentation, it is the height of 80s usability. Note that we are looking at the top of a menu tree; on the bottom two rows are a *secondary menu* that changes as the cursor slides through the choices, as well as a fixed menu of F-key functions. These F-keys, fur-

ther, refer the user to one of the two main modes of help that characterized the period: a "Help Index" that opens up an entire on-line manual, searchable by topic and, in addition, context-sensitive help that provides information appropriate to the position of the cursor at the time of the request. The user's actions consist in sliding the cursor bar and selecting. There is no typing until we reach the farthest branches of data entry.

The interfaces for such systems are clean, clear, and rectilinear. One moves up and down a hierarchical information stack; even with a mouse, there is no "dragging" or "rubberbanding." They resemble orderly kitchens and workspaces, with all the tools put neatly away, except for a simple sheet of instructions and a sign pointing to the Exit.

The most extreme example of this style, of course, was the ubiquitous opening screen of WordPerfect 5.0 for DOS. Through ingenious design, hundreds of word processing functions were converted to menu trees, each hanging from one of 40 combinations of the F, Shift, and Control keys. The interface was an all-blue, ultra modern, Bauhaus-style kitchen, where all the tools were hidden away in drawers without knobs or buttons. Even when the program offered a menu bar and pull-down menus (Version 5.1), many users continued to use the smooth uncluttered interface instead (as they still do, by the millions, today).

Supporting such stable, reliable business and administrative software also fostered a kind of standardization in user documentation. Manuals became illustrated lectures, with the bulk of the book containing captured or simulated screens with explanatory captions. There were, of course, stylistic conflicts. Some writers still wanted to describe and explain each menu and screen as a separate entity, arranged alphabetically (product-oriented), while others wanted to link the screens into meaningful scenarios (task-oriented). Generally though, for this type of technology *the problem of the users' manual was solved*. A good manual would contain a useful sequence of representative

screens, filled with realistic data, along with enough exposition to explain what was not obvious from the illustration.

Protectiveness and Paternalism

This quest for usability may be regarded as a paternalistic imposition of fixed options and inflexible rules *for the protection of the user*.

Thus, what made this form of application, as well as its documentation, possible was the notion that the responsibility for the appearance of the screens, the operation of the keys, and the sequence of steps required to complete a transaction was with the *developers* of the system. Unlike the old days of the 60s and 70s, with their empty, structureless interfaces, in which the job of the developer was merely to provide the user with tools and resources, the new era of usability meant that the developer or programmer had to work everything through in advance, even to devising a table for context-sensitive help that would predict what was perplexing a user at an impasse. Developers, at the urging of writers and teachers, were forced to become paternalistic about software, although many never really wanted to.

Users were, of course, being protected from themselves. It was argued, convincingly, that if users felt they could not make a bad mistake while working on their systems, then they would work with more energy and be more willing to use trial-and-error as a way of solving problems. [4] Similarly, the 1980s pursuit of "usable documentation" may also be interpreted as the development of paternalistic writing and publishing techniques that would protect readers from themselves.

Until the documentation of business software became a large minor industry in the 80s, the presumed relationship between writers and readers of manuals was very much like the traditional relationship between literary authors and their readers. The writer's responsibility was to write clearly and correctly and to "provide information";

if the information was wide and deep enough, we called it “comprehensive,” [5] our highest rating. And the reciprocal part of this notion was the expectation that readers would be intelligent, resourceful, patient; we expected them to follow instructions (“How to Read This Manual”) and to remember what they had read from session to session. Until the 80s, we expected the manuals to be somewhat more communicative than the interfaces they supplemented—but not much. At the very least, everyone, writers and readers, did *not* expect manuals to be easy or “intuitive”; moreover, failure to read or use one’s manual was taken as a sign of laziness and irresponsibility on the part of the reader, not, as was later the case, an indictment of the quality, the “usability,” of the publication.

By the mid-80s, though, unread or unused manuals were more likely to be blamed on bad writers than shiftless readers. (This was the same era in which the idea became fashionable that failing students are really evidence of failing schools and teachers.) The nascent software documentation profession adopted paternalism as its model, taking upon itself the responsibility for the “usability” and “readability” of publications. It was not uncommon for those young and ambitious writers who came of age in this decade to imagine manuals that could be so well put together that even the least sophisticated and most reluctant readers would find them easy and helpful. In contrast, many of the technical writers who reached maturity earlier, in the 60s and 70s, were outspokenly annoyed about this preoccupation with the special limitations of novices, and rather obtuse novices at that. [6] This friction between the generations of writers did not really bother the young writers much, though; in truth, the intellectual leadership of the new writing era was to be found in human factors psychology (usability testing, visual design...) and in instructional technology, rather than in the old texts on technical writing.

Paternalistic writers (like me) even expropriated the terminology of software engi-

neering in general, and user-friendliness in particular. My book on the subject [7] claimed that, although writers could not coerce readers to read any words in any order, the writer’s main objective should be to eliminate those well-understood factors that distract, mislead, derail, or overburden the user. The central goal was to “engineer” the document so as to shape and control the attention of the reader.

What the profession learned about documentation, and applied with considerable success, was *how to reduce reader overhead*, the effort needed to find the right words and pictures and put them in the right order. (Notice again: Usability consists in eliminating unproductive choices; the less freedom the readers have, the less likely they are to misread or misuse the document.)

And, as a fortunate coincidence, desktop publishing technology gave even the lowliest writer in the smallest company immense new powers to determine the final form of documents and to apply what was known about page design and typography to further control the attention of readers. Information developers benefited as much from the new computer technologies as any other profession, and more than most. Whereas in the early 70s the typical technical writer wrote in pencil on a lined tablet and depended on someone called a “repro typist” to prepare the final, “camera-ready” document, by the mid-80s writers were making the pages directly, with almost too much graphical and typographical freedom.

Again, this technology enhanced their ability to make the writing accessible and readable. The most junior writer could now deliver publications with comfortable column-widths and readable typefaces. To reduce the amount of page flipping and to simplify the task of editors, we could now integrate text and illustrations seamlessly, something that only professional print shops used to be able to do. Ironically, in the 70s, one of the pleasures of being published was seeing our words typeset and justified; by the late 80s, most savvy documentors knew that text-

justification was a liability and were producing ragged-right texts on purpose. To most people, this merely meant that the average computer manual was “prettier” or more professional looking; to the professional, these documents were designed for readability and produced genuine improvements in reliability.

With usability—that is, ease of learning and use—as its rallying call, the profession advanced dramatically. A comparison of the Lotus 1-2-3 manuals of the early 80s and late 80s, for example, shows what appears to be centuries of progress. And, indeed, the remainder of this decade will probably be spent disseminating the advanced techniques of usable documentation more and more widely, until well-made, good-looking manuals are a commonplace.

Throughout this decade, very few have challenged the underlying precept: that the user is like a dependent child, in need of constant parental protection from the dangers of the world.

The Silent Retreat from Usability

Somewhere around 1990, with the advent of universal graphical computing, the goal of usability—that is, ease of use—was unostentatiously downgraded and superseded by new goals. Ironically, the tactic that enabled this change of policy to slip into the tent unnoticed was that it masqueraded itself as the latest frontier of usability and “friendliness.” (Indeed, nearly all developers continue to talk about ease of use as though it were still the prime objective.) Especially given that the Apple Macintosh organization had always insisted that its graphical style of computing—“intuitive,” according to its enthusiasts—was markedly easier to use than the DOS or UNIX or MVS styles of computing, it seemed only logical that the non-Macintosh world could substantially increase its ease of operation, merely by adopting a graphical interface of its own.

And, of course, there is one sense in which the transition to Windows and windows-like

operating systems has made life easier: After the many days it takes to learn how Windows works, there is a single repertoire of conventions and a single familiar operating shell that does not have to be relearned with each new application. Of course, much of the same result could have just as easily been achieved in the character-based world with, for example, the Lotus menu shell, used so effectively in the character-based versions of Lotus 1-2-3 and Lotus Freelance.

What is too often overlooked is that the widespread appeal of graphical computing derives not from its ease of use but from other benefits. What most draws us to Windows and such is, first, the **power and versatility** of the shell, which, among other things, provides several ways of doing nearly every task. Indeed, the current widely used version of Windows (3.1) is so complex, undisciplined, and unpredictable a system—especially regarding its protocols for installing new applications—that one depends on these multiple paths. Often, what should work does not and, in these instances, experienced users have learned to be grateful that when one path fails, there is always another way. (If we can't insert it as a file, insert it as an object; if we can't insert it at all, we copy it into the clipboard and paste it; if the pasted bitmaps make the file too large, we convert them, etc.)

This sense of endless possibilities, including the knowledge that there are many undocumented techniques, gives Windows sessions their peculiarly adventuresome mood and style. And this sense of challenge, intellectual frustration, and inventive problem-solving, although it may be characterized as ultimately “useful” (in the way that a language with more vocabulary and richer syntax is more useful in communicating than a language with less vocabulary and simpler syntax), is diametrically opposite the prevailing utilitarian notion of the 80s' user-friendly menu-tree!

Feature-richness, feature-abundance, is another part of the appeal of post-1990 software. Even though windows-type shells

ease switching between applications, increasingly we have loaded up our applications with so many features and utilities that one need not leave them to do other tasks. A 1994 word processing system, for example, uses the "frames" that were once the province of desktop publishers; most include calculators, simple spreadsheet and charting programs, templates for making mailing labels, a simple drawing program (with the ability to import all standard artwork), automated routines for form letters.... So-called spreadsheet programs include filing systems, clip art libraries, presentation software, and a few sound effects.

Feature-richness, far more than price, seems to drive the market. (Indeed, the ability to use the same graphical conventions to launch an ever-larger set of functions must have encouraged developers to so load their applications with features that the resulting complexity sometimes offsets the usability benefits of the graphical operations.) Consumer and business magazines make side-by-side comparisons of competing products and buyers vote against those that lack even the most arcane features. Mighty software companies sometimes offer stripped-down versions of their products, only to discover that consumers prefer the bloated ones. (For example, even though most users never produced anything more ambitious than a letter or memo on WordPerfect 5.x, the leaner, less expensive, less resource-hungry LetterPerfect version never appeared on any of the software best-selling lists.)

Feature-rich programs bedazzle the novice and also tend to be buggy, slower, and unreliable. Information services offer patch programs; major companies issue invisible upgrades without changing version numbers; bulletin boards fill with problem messages. Although some of this poor quality is due to the rush to market, most of it is inherent to the feature-richness itself. *It is an axiom of software engineering that when a product exceeds a certain level of complexity, it is no longer possible to test all of its uses.*

Moreover, when two feature-rich programs interact in the same session, mystical errors often occur in the system. Feature-richness introduces additional menus with additional tiers of decisions. It opens dialogue boxes containing 10 choices instead of one. It can make the low-level task of saving a file into a high-level decision problem. Why is it the apparent goal of every software developer?

What has emerged in the last three or four years as the marketing equivalent of user-friendliness is **user-customizability**. Apparently, no product or application will be well received these days unless it offers a wide range of options to users: colors, panel sizes, opening screens, icon displays, nomenclature, warning policies, interruption priorities, cursor styles, sounds, links with other programs, exit sequences. Users, it appears, demand the right to change the width of help panels as well as the color of the highlighted text; they insist on being able to change the size of the screen text, even the number of lines of resolution; they not only want to pick their colors, they want the ability to switch "palettes" between jobs, trading off beauty for speed if they feel like it. Why, some even want to be able to convert a busy Windows-based word processing screen into the pristine blue of WordPerfect—a feature provided by, of all people, Word for Windows.

Others want "multiple desktops," in which they cluster these personal choices into stylistic profiles that can be invoked as the occasion or mood requires. And some even demand the right to have fun, to intermingle humorous sound effects and clips from movies and TV shows among their screens. *All these baroque indulgences are inconsistent with the classic notion of user-friendliness: a few, well-defined unambiguous options in any situation.*

Complexity versus Usability

Feature-rich software is, by any traditional engineering measure, less usable than software that does one or two things simply and well. In analytical terms, features and op-

tions add decision nodes, which in turn increase the number of paths through any process or subprocess. And the number of paths through any process is the best inverse predictor of its reliability.

This path complexity affects the software itself, which occasionally finds itself caught in a maze or loop from which it cannot escape, uneven able to issue some portentous error message. But it more deeply affects the users, who, at every stage, have an opportunity to make a mistake or, worse, ask the software to perform in a way that will fail.

Again, path complexity lowers reliability in two fundamental ways. First, it increases the chance of a logical impasse; second, it reduces the chances that the impasse will be tested. Every yes-no decision node added to a procedure doubles the number of tests needed to validate that procedure. And when there are 3 or 4 or 10 decisions (like colors), the multiplier increases accordingly.

The need to reduce complexity, to minimize the number of paths and limit the depths of procedural nesting, has been appreciated for decades; it is one of the foundations of software engineering [8]. What is not surprising is that the same principle is emerging in "usability" research. Shriver and Hayes [9] are among many who have researched the relationship between the complexity of a problem task and the likelihood of success. Their recent finding is that the number of "plausible alternatives," that is, the number of feasible paths at any moment, is one of the best inverse predictors of success!

The new software, however, is quite different from the stereo and video components that the Shriver and Hayes subjects are trying to connect. In most cases, there is only one correct way to achieve the objective. What saves the Windows user, in contrast, is that, although one has scores of choices at any moment, *several are likely to work*. There are a half-dozen ways to start a program, for example, even without add-on software that lets us start the programs with special buttons and pads. And within the

programs, simple chores can be performed in so many ways that most users never even try them all.

In the popular Word for Windows, for example, there are at least 13 ways to highlight a single word. And, to make matters even more interesting, there are about 11 more techniques for special cases. These 24 options moreover become at least 48 because the cursor keys or other "shortcut keys" can replace some of the mouse functions in these 24 options. Then there are ways to program the right mouse button, not to mention third-party software with additional options.

And there are still other paths in the form of macros, which in turn can be activated through pulldown menus, or by adding menu items to the standard menus, or even by creating a new icon which can be assigned to any of the several toolbars, which can be in any position on those bars, and whose insertion may require the user to choose which existing button to delete, or, alternatively, to create specialized toolbars for various modes....

Is it any surprise that, despite all the talk of intuitiveness, these new products ship with immense manuals?

And what argument can be made to show how this luxuriant array of options contributes to usability, either in the narrow sense of ease or the broader sense of useful? Does it make software more usable when we triple the resources needed to store it or double the time needed to open and close files? Does it help users to provide them with swollen, intimidating manuals, when they have already shown themselves unhappy with smaller, simpler ones? Does it aid efficiency or productivity when pull-down menus provide 100 options? And why, in spite of the obvious answers to these rhetorical questions, do most users prefer the kinds of products implied in these questions and insist that they are more usable?

The New Paradox: GUIs "Feel" Friendly

This paper claims that the new software, as typified by the latest generation of products from Lotus Development and Microsoft, is harder to use than the analogous products of the late 80s. The new interfaces are filled with cryptic icon images (not intuitive), and the one-word names on the menus are often no help in choosing, for example, where one converts a portrait page to a landscape one. Every new program contains, we strongly suspect, 100s of features that we never discover by trial and error. How would one learn that menu choices followed by an ellipsis (...) lead to a dialog (sic) box? Or that Alt+F4 functions like the Escape key in DOS? Would one ever discover the extremely useful "Spike" option in Word for Windows without learning about it elsewhere?

The new manuals, beautifully written and illustrated though they are, are most often feature-by-feature, object-by-object demonstrations in the tradition of DOS manuals, with dialog boxes replacing the command syntax. And, in most cases, they are longer and more intimidating than ever.

Nor is the behavior of pointing devices intuitive or easy. The relationship between the mouse and the cursors (several modes in each program) is subtle and must be learned (and of course can be customized and personalized); the logic of trackballs is even less apparent and considerably more error-prone. Although the new software generally allows us to use large-type screen displays, these applications are nevertheless very hard on the visually impaired, as are pointing devices on people with injuries and handicaps.

And yet, when one claims in public that the new programs are harder to use than their predecessors, especially before a group of technical communicators, the reaction runs from skeptical to contemptuous. Unfortunately, the ease-of-use sense of usability is the central evaluative criterion for computer technology in many minds, so that saying

that one product is *easier* than another is equivalent to saying it is *better* than another. And since everyone (including this writer) thinks that the new software is better than its predecessors, how can it also be less usable, harder to use?

The answer is that we have replaced a simplistic model with a more interesting one. In the 80s, we thought of learning and using software as an essentially unpleasant task, putting uncomfortable burdens on the memory and dexterity of the user. The users were fearful and insecure, often reluctant participants, most of whom would not use computers if they could avoid them. (In contrast, people who actually loved to work with computers were frequently marginalized with condescending names like "tekkie" or "hacker" or "nerd.")

Our older models for training and documentation were industrial. The user was like a production worker, ignorant and without intellectual resources, fearful of errors, always grateful for the easiest, least stressful way to work. In this context, training, documentation, and then interface design, were all meant to reduce stress and protect this disaster-prone user from pain and embarrassment.

This simplistic model equated painlessness and the reduction of stress with pleasure. People with easy-to-use systems, it argued, are happier than those with hard-to-use systems.

The newer, 90s model suggests that pleasure may sometimes come from ease, but it may also come from power, flexibility, ornamentation, indirection, personalization, challenge, surprise, creative tension, and even danger. For example, the old usability model said that giving users lots of typographic options would only confuse them and encourage them to fritter away corporate time on frivolous formatting games; the new model says that having hundreds of typefaces makes life more interesting, especially when one decides that there are too

many and chooses one of several ways of managing the problem.

To illustrate: While composing this manuscript I have used a program that automatically inserts true quotation marks in place of the inch-marks in the basic ASCII character set. From time to time, however, the software (for reasons no one can explain) places the initial quotation mark too close to the initial letter, so that they overlap. This problem occurs at unpredictable intervals and with no apparent cause. Rather than abandon the use of these SmartQuotes, though, I merely highlight the offending pair of letters and adjust their kerning with another utility in the same program.

The old utilitarian model would have discouraged developers from offering the user this essentially decorative feature in the first place and, if it proved buggy, would have encouraged them to remove it from the software or delete all references to it from the documentation. After all, why should a business professional be obliged to learn about kerning and leading?

In the old model, in the era of the character-based terminal, the user looked through the screen to see other things; the system was a conduit to non-system things; the best technology was "transparent." In the new model, the user looks at the screen [10] sees the text and the colors as opaque graphic entities, is aware of the links to other screens, is fully involved [11]. There is a sense of both work and play.

This is not to say that today's designers are uninterested in usability. It is still possible, after all, to make modern interfaces look like the austere screens of the dumb terminal days. But today, most usability engineering is within paths, objects, and subprocesses: task by task. There are more and more routines that run automatically (that is, after only one selection), and many more opportunities to rehearse and preview complex decisions. There is also ongoing basic research, of the kind that, according to reports in the popular PC press, persuaded Lotus

Development to replace the word *options* with the word *preferences* on several of its menus. And there is an abundant amount of what used to be called "help," but which now turns up in a dozen sites (more on this below).

Developers continue to reengineer their systems to share common modules and reduce overhead. Faster devices and cheaper storage make complex tasks go faster, making it more attractive to revise and retry our projects.

In general, though, each wave of innovation is more likely to make our technology more interesting, stimulating, engaging, and flexible—what could be called pleasurable—rather than easier.

The New Users

These new design objectives—power and versatility, feature-richness, and customizability—presume a resourceful, inventive new user, one who not only wants to turn the ignition key but also to look under the hood and tweak the timing [12]. This putative user is neither fearful, error-prone, computerphobic, nor feckless. This user is not the hesitant neophyte imagined at the height of the usability epoch, and does not depend in the same way on the protective intervention of the interface designer. Increasingly the real user of today's computer technology may be someone who has been studying through a hypertext utility since grade school. Eventually, then, most of the usability-thinking of the 80s will be invalid—if it is not already.

Until recently, an essential ingredient in "audience analysis" was differentiating between the novice and experienced user or, more important, the clerical and creative user. The clerical user looks through the screen to the simple task of converting data from one form to another; the creative user looks at the screen, creating panels, frames, and pages of information. One might properly criticize a company that tried to support its clerical workers with documentation

suiting for creative users, such as by providing an elaborate hypertext help file for someone who merely needed a table of allowable entries for a given field.

But are these clerical users of the 90s the same as those of the 80s? Are the airline reservation clerks, supermarket checkout clerks, main lobby receptionists, customer service agents, postal workers, library aides, doctors' receptionists and admissions clerks, life insurance sellers, newspaper circulation clerks, delivery truck drivers...are most of these the trembling computerphobes of the 80s? Or are they not more likely to be people who have manipulated complicated video games and used computer programs ever since grade school?

Even in offices where the PCs are linked through a server, a practice that should standardize the interfaces, one rarely sees identical displays at two stations. Everywhere one sees that workers we once presumed to be dependent and error prone are also lured by the attractiveness of complexity and customization. At the very least, this new-model user is ascending. A decade ago, Alan Kay wrote "the protean nature of the computer is that it can act like a machine or like a language...." [13] In effect, nearly everyone who works with computers, including those that do clerical tasks, is coming to think of computer technology more as a language. Consequently, our way of supporting must also change. Kay observes further:

Computer literacy is not learning to manipulate a word processor, a spreadsheet....

Computer literacy is a contact with the activity of computing deep enough to make the computational equivalent of reading and writing fluent and enjoyable. As in all the arts, a romance with the material must be well under way....

Since about 1990, the romance has flourished.

The Decline and Fall of the Manual

With few exceptions, the users' manuals accompanying the new products have grown ever larger. (For example, the manual for PC Tools for Windows—a product that permits the user to make hundreds of additional personal customizations in the already customizable Windows shell—includes an attractive but largely incomprehensible manual of nearly 1000 pages.) And the bookstores are filled with third-party "made easy" publications that are even more immense.

Despite advertising claims to contrary, it is reckless to tackle any major software product without the user documentation, unless one limits oneself to the subset of features that are familiar and intuitive. One will almost surely never stumble on the already-mentioned "Spike" feature in Word for Windows without reading about it; and one will almost certainly never be able to use the same program's delightful "style paintbrush" without reading the instructions two or three times.

Today, though, as in the days of the simplified interface, most people still disdain and abjure manuals, no matter how well written. Paradoxically, even though feature-richness and customizability have all but murdered the notion of intuitively clear software, nevertheless users now believe, almost universally, that applications should be learnable without books. Even though the products from most major developers are shipping with larger and larger manuals (which they genuinely need), even though most customers would feel cheated if they didn't get these compendious collections, even though one of the few reasons that people buy their own software (rather than copying a friend's) is that they want the documentation...despite all these, most users still consult a manual only when they are in trouble, and even then with annoyance. And the users included here are not just neophytes and people with limited reading skill, but nearly everyone.

(This author, for example, solves most software puzzles by sending out an appeal for help on the appropriate bulletin boards; the solutions come quickly and are usually sound. This, by the way, is the best method for solving problems caused by the interaction of two programs—as when Hijaak interferes with Word's graphics filters. And it is the quickest way to learn about bugs and fixes in popular products.)

A second assault on usable documentation has been the **nearly universal preference for online help**. But the style of online help is not the lean, context-sensitive screen that usability experts developed in the mid-80s. Rather, when today's users ask for help they are led to the Table of Contents or Index of a node-rich hypermedia library, which they can search until their problem is solved. And it is not uncommon (as with Lotus Freelance, for example) for these help files to contain better and more recent information than the manuals.

Hypermedia help, like that produced with the Windows Help Engine, is diametrically opposed to the notion of easy-to-use help screens. The typical implementation is filled with dense text paragraphs, in a hard-to-read typeface. When users have chosen to "maximize" the window, the rows may have over 120 characters in them—unacceptable from a paternalistic, human factors perspective. Again, hypermedia help presumes a resourceful, nearly intrepid user, who likes to read, who reads well, and who knows how to use a shallow index. This is not the timid, artless user defended by the professional documentors of the 80s.

Moreover, just as today's users expect that there are always several paths to an objective, they also expect that help can be found not just on help screens, but everywhere! Fields expand; balloons and menus are activated by the second mouse button; coaches and wizards come to rescue; status bars give prompts.... (And, of course, the options are customizable and there are also add-on products to provide an even richer assortment.) The functions that used to be local-

ized to help screens and manuals have now been balkanized throughout the domain of the application and operating shell, to the point where there is rarely a clear distinction between what is the program and what is the support and documentation.

And there are other threats to the power and influence of manual writers. **Customization of interfaces and applications has gradually eroded the writer's ability to create task-oriented documentation.** If no two users have the same screen, then what should appear in the manual? The answer: Objects, menus, and dialog boxes, explained discretely. Why, in the current epoch of graphical shells, there are so many ways to do anything, that the writer cannot even discuss them all!

"Minimalism" was, in some ways an early appreciation of the fact that step-by-step scripted applications are the only kind that lend themselves to the utilitarian conception of "task-oriented" documentation. Yet, in much the way that many believe a good system must be "easy," others believe that good documentation must be task-oriented and continue to describe excellent minimalist publications as task-oriented, even though the term fits badly [14]. Ironically, writing manuals for Windows applications feels more like documenting the blank-screen applications of the early 80s than the menu-driven applications of the late 80s. Consider the two opening screens for the current version of Quicken (Figure 2).

Absent the ability to control, or even predict, the look of the screen and the sequence of processes, the documentor is forced to describe the objects, the process molecules, and hope the users will make sensible strings of them. Rather like a DOS reference manual!

And the issue is not the familiar 80s paradigm in which the technical communicator produced instructions for the standard, plain-vanilla version of the application and offered more difficult documentation for those who wished to customize or enhance the basic version. Although today's Win-

dows applications often give the user the choice to leave out features and components at installation, they still tend to open with a dazzling array of options. And, unless instructed otherwise by the users, many applications "recall" how they looked when the users closed them and, thus, reopen with a different initial screen every time they are used.

Users apparently want the right to control their screens and, increasingly, they want the right to make their own virtual documents. The writer's notion of what readers need and in what order, the writer's skill in designing pages and displays that reduce the reader's burden and increase the chance for successful communication, the writer's continuing quest to guide readers through difficult paths of instruction...these contri-

butions by the writer are frequently discounted these days. Indeed, the idea that writers should *not* determine the stylistic and formatting conventions of their own pages is one of the core ideas of SGML [15]. The objective of breaking the link between a writer's words and the specific hardware and software needed to present those words exactly as the writer intended is one of the main goals of the military's CALS programs [16]. In short, our attempts to teach and help, to protect and reassure, are often viewed as irrelevant intrusions.

Beyond User Documentation

Although we have been in the post-usability era since about 1990, there is little awareness of this new paradigm among working writers. Indeed, ease-of-use has been the central organizing principle in most of the university programs for computer industry writers, and those eager graduates are just now entering the workplace.

As often happens, the history of the documentation profession is paralleling the history of the programming profession, only a few years later. Although each important development in software engineering has obviated many of the practices of the previous epoch, working programmers generally continue to pursue the old goals with the old methods. Even today, there are thousands of them making awkward fixes on ancient COBOL programs; they have no idea that programming these days is mainly the manipulation of software objects! And just as most programmers demonstrate no apparent understanding of the structured methods of the 70s, the CASE techniques of the 80s, or the Object-Oriented style of the 90s, similarly most working documentors have barely advanced to the modular, engineered manual and are still composing 1960s-style artistic works in their cubbyhole garrets.

In the mid-80s one often encountered resistance from professional writers when they were urged to abandon their "artistic stereotypes" and to chunk their manuals into

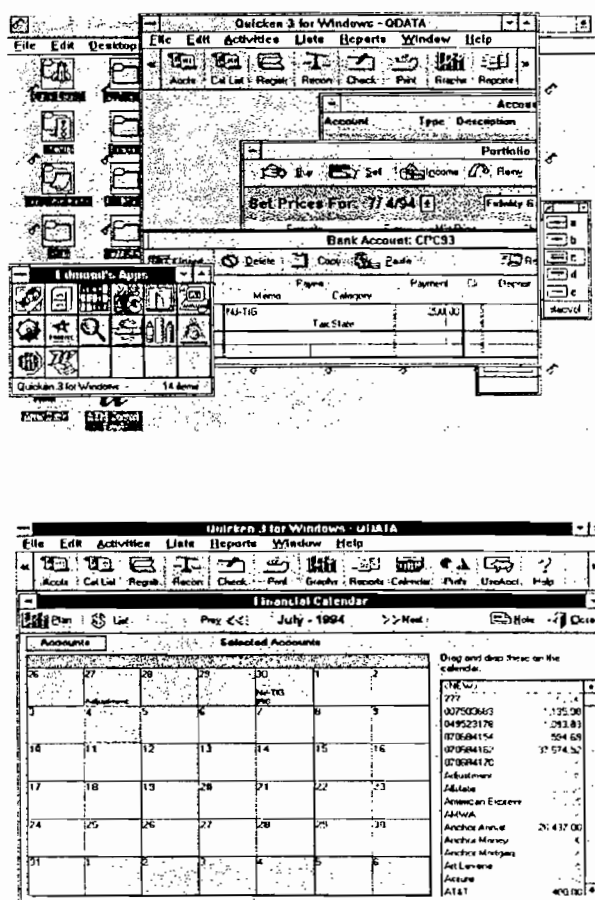


Figure 2. Two very different opening screens for the "same" application

small modules, reusing the modules whenever possible. Often they would bristle at the assertion that manuals written entirely by one person working in privacy are inherently unmaintainable.

The alternative (as proposed by me and others) was an engineered manual, designed by a team of senior writers and developers, tested in a storyboard, and then "implemented" by a variety of people writing small, well-defined pieces of the document. In general, traditional technical writers resisted this model, mainly because it threatened the intellectual independence and pride of authorship that, for many, were the best parts of the job.

But the emerging 90s culture for documentation makes this old "structured documentation" of the 80s seem pre-Raphaelite by comparison. First, it is extremely difficult to provide "task-oriented" user documentation for systems and applications that are more like languages than machines. (That is, for systems which are more like a collection of useful "agents" than a well-defined, procedurally stable application.[17]) Second, most users wouldn't read our manuals anyway.

If not from a practical perspective, then from an historical one, we have reached a moment when *nearly all documents will be virtual documents, extracted by users from information-rich databases*. Whether we call the technology SGML, or CALS, or just Help, the effect is the same: a recasting of the role of the information developer from a protective advocate and teacher to a provider of information objects [18].

This new communicator feeds small units (chunks, modules, articles, objects, panels) of information into document databases. Although these units will be, to the extent possible, self-contained, they will also carry logical markers and semantic links that connect them to other units, eventually in all other databases. The pressures upon these writers will be to keep the units small and to strip them of any formatting restrictions: typeface, font, columns, page breaks.... The

goal will be to enable anyone anywhere interested in the content of the unit to be able to reach it and to read it (display, copy, print, speech-synthesize), irrespective of the software and hardware on the user's desk.

In this scheme, the technical communicators will find it difficult to act as protectors or teachers, for many a favorite role. Except within the tiny domain of the unit, they are not to control the order in which readers read, the pace of the presentation, and especially not the look and feel of the pages and displays. One even hears military enthusiasts hope that the information in the data base will be the raw byproduct of engineering, rather than something that has been interpolated by a technical writer who, of course, cannot anticipate what the user needs to know.

This new comprehensiveness returns us, in effect, to pre-usability expectations. The future documentation must be exhaustive and richly tagged with logical connections, because writers may make few decisions about its relative usefulness or appropriate sequence. In fact, as our systems get ever faster and more powerful, we can expect that it will no longer be necessary even to worry about system overhead and that, therefore, there will be no need to limit the number of hypertext links and paths through the online information. (Every word a node!)

Moreover, whenever possible we will reuse existing document objects with new local data, so that even the creation of these documentation units will be reduced to filling in the blanks. And thus, eventually, more and more of the units will be generated automatically.

So, we are entering an era in which, instead of restricting themselves to finding ways that make the users' experience simple, safe, and free from difficult choices, technical communicators should be thinking of ways to make software more stimulating and satisfying: perhaps producing the software for themselves instead of acting in the second-

dary or "support" capacity. The issue is no longer just ease of use, nor even usefulness;

the issue is the quality of human-computer engagement.

References

1. Paul Booth, *An Introduction to Human-Computer Interaction*, London: Lawrence Erlbaum Associates, 1989
Jeffrey Rubin, *Handbook of Usability Testing*, New York: Wiley & Sons, 1994
2. David Leonard and A. Lynne Waller, "Usability Planning for End User Training," *SIGDOC 89 Conference Proceedings*, Association for Computing Machinery, 1989, pp. 137-142
3. Philip Rubens and Brenda Knowles Rubens, "Usability and Format Design," in Stephen Doheny-Farina (ed.), *Effective Documentation: What We Have Learned from the Research*, Cambridge: The MIT Press, 1988
4. Trial-and-error, with rapid feedback, is the basis of much minimalist documentation and training. See J. M. Carroll, *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*, Cambridge: The MIT Press, 1990
5. E. H. Weiss, "Comprehensive User Documentation," *Auerbach System Development Monograph*, Auerbach Publishing, 1981
6. In private conversation, Sandra Pakin, editor of the *Journal of Documentation Project Management*, has expressed to me the opinion that the documentation profession has become excessively concerned with the needs of novice users, who, she points out, are usually a minority of those to be served. At another time, Joseph Chapline, author of one of the first users' manuals (for the BINAC) and recent winner of the ACM SIGDOC Rigo award, has expressed the concern that too many manuals are written for the "ultrafeeble" user, at the expense of the more typical user.
7. E.H. Weiss, *How to Write a Usable User Manual*, Philadelphia: ISI Press, 1985
E.H. Weiss, *How to Write Usable User Documentation (2/e)*, Phoenix: Oryx Press, 1991
8. A cogent treatment of this idea can be found in T.J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, SER-2, no. 12, December 1976
9. K.A. Schriver and J.R. Hayes, "The Impact of Product Complexity on Using Consumer Electronics," *Performance Improvement Quarterly...*(1994)
10. This *through-at* distinction is a recurring theme in R. A. Lanham's *The Electronic Word: Democracy, Technology, and the Arts*, Chicago: University of Chicago Press, 1993
11. Lanham (op. cit.) points out that Marshall McLuhan's discussion of how people are affected by television is also a remarkably accurate description of people interacting with a computer. From *Understanding Media: The Extensions of Man*, New York: McGraw Hill, 1965, p. 336:
For people long accustomed to the merely visual experience of the typographic and photographic varieties, it would seem to be the synesthesia, or tactual depth of TV experience, that dislocates them from their usual attitudes of passivity and detachment. ... TV is above all a medium that demands a creatively participant response.
12. In Robert Pirsig's *Zen and the Art of Motorcycle Maintenance* (New York: William Morrow, 1974), there are two kinds of cyclist. The first wants to turn the key and have the engine start and run reliably, without knowing why or how; the second kind wants to tear apart the machine frequently, maintain it, improve it, and know why it performs as it does and how it can be persuaded to perform better. Until recently, business users of computers were presumed to be in the former category...especially by most technical communicators.
13. Alan Kay, "Computer Software," *Scientific American*, Vol. 251, No. 3, September 1984, p.59
14. Clinging to "task-oriented" as a term of approval, but recommending something rather different, an anonymous referee for this paper writes (November 1994):

Documentation has to go beyond GUI details and describe TASK objects and actions.... A screen-specific play-by-play may seem comforting, but it's not the best way to promote longterm learning!

15. International Organization for Standardization (Geneva), *ISO 8879* (1986)
C. Goldfarb, *The SGML Handbook*, New York: Oxford University Press, 1990
 16. Military Handbook, *Department of Defense Computer-Aided Acquisition and Logistical Support (CALS) Program Implementation Guide*, MIL-HDBK, Dec. 1988
 17. This conception of the "agent" appears in Marvin Minsky's *The Society of Mind* (New York: Simon and Schuster, 1985); an agent is any part or process of the mind that by itself is simple enough to understand – even though the interactions among groups of such agents may produce phenomena that are much harder to understand. (p. 326)
 18. E. H. Weiss, "Of Document Databases, SGML, and Rhetorical Neutrality," *IEEE Transactions on Professional Communication*, Vol. 36, No. 2, pp. 58-61, 1993
-

Seminars, Courses & Speeches

Business/Professional Communication

How to Sell in Writing (Proposals & Business Cases)

The most important business writing is the *advocacy document*, the pitch for funds or approval.

- Analyzing your audience and Win Strategy
- Presenting the “case” with logic and persuasiveness
- Using business graphics to demonstrate and prove

How to Write *Globally*

International business requires sensitivity to the language, culture, and expectations of the international business partner.

- Editing for clarity and readability
- Screening for figurative and idiomatic confusion
- Designing accessible layouts and appropriate feedback paths

Final Draft: The *Especially* Clear Sentence

Good writing is *rewriting*; only revision can assure clarity, correct tone, freedom from errors, and readability.

- Emphasis and making your point
- Twenty flaws in first-draft sentences
- Style-checking software: Can you trust it?

The Art of the Pitch

A well-made presentation is a small five-act play, where each element contributes to effectiveness.

- Strategic planning and design
- Managing stage fright
- Using PowerPoint™ and other presentation tools
- Handling questions and objections
- Creating useful handouts

The Art of Effective E-Mail

To use e-mail well, the writer must exploit its strengths and adapt to its limitations.

- Attention-getting subject lines
- E-mail style and grammar
- Discipline and etiquette for e-mailers
- To attach or to embed ...

Technical Communication

A Writing System for Technical Professionals

Technical professionals cannot achieve their professional goals unless they write their correspondence, reports, and documentation with power and precision.

- Creating documents as engineered information products
- Eliminating common errors and time-wasters
- Writing for *nontechnical* readers

Preparing English Tech Documents for International Readers

Although customers and clients around the world read English quite well, it is still necessary to edit international technical information for the E2 reader.

- Making documents *culture-free* and *culture-fair*
- Correcting problems of style, idiom, and syntax
- Using controlled English
- Adapting to local sensitivities and cultures

Effective Quality Manuals/ Usable Procedure & User's Manuals

A manual is a device that supports people in their work; when well designed, it teaches procedures, enforces standards, and saves money.

- Documenting ISO 9000 and other quality standards
- Replacing unreadable and unmaintainable prose with scripts, tables, and diagrams
- Testing for usability and enforceability
- Designing modular, maintainable publications
- Storyboarding and project management

The Craft of User Requirements & Functional Specs

Those who use information technology and those who create or acquire it must communicate their needs and expectations clearly, especially at the beginning of the design cycle.

- How **User:Developer** communication fails
- Beyond the Waterfall Model
- Tools and processes for functional specification

Organizational Communication

Meetings that Work

Meetings should be energizing and productive—never boring or a perceived waste of time.

- Objectives and agendas: staying on message
- Two warring cultures: ratification vs. exploration
- Roles and games played by participants
- Secrets of master facilitators
- Cultural variables in international meetings

There's Only Now: Managing the Professional's Time

Despite the array of electronic time management tools, too many professionals feel overworked, stressed, and never quite on top of their work.

- Attitudes about time
- Five immutable rules of time management
- Time management traps and how to avoid them
- Products and tools and how to choose/adapt them
- Getting long-term goals and projects into your short-term calendar

Raising Culture Consciousness

An urgent need for international business professionals is to learn, and adapt to, the culture of the communities or countries where they wish to do business.

- Dimensions of difference
- Context and communication
- Individualism versus collectivism
- Timing and pacing (the hidden dimension)

Turning Words into Money: Business Plans & Cases

Projects need funding, capital; even the best ideas can fail for lack of a convincing business plan/case.

- What impresses funding sources
- Missions, visions, and goals
- The logic of the 'business case'
- Clear, persuasive language and graphics
- Presentations for executives and sponsors

Speeches/Short Programs for Professional Gatherings and Meetings

- How to Sell an Idea** Why won't people follow your advice? There are eight barriers that keep us from accepting new plans and approaches... and specific techniques to overcome them.
- The Secret of Professional Fulfillment** The key to mental health and productivity—on the job or at home—is *equilibrium*: keeping all of life's eight competing values in balance. The tendency is to neglect some while pursuing others, a practice that leads to anxiety and alienation.
- Re-Inventing the Memo** Do you have trouble getting your point across to co-workers? A memo is NOT a work of literature, but, rather, an engineered product, designed for clarity, power, and speed. Twelve tactics increase the chance that a memo (or an e-mail) will be read.
- The Odor of Mendacity—Why People Don't Believe You Anymore...** In school, we learn ways to "improve" the truth by puffing up our writing with words that inflate, obscure, and disguise. Business and professional speech and writing are filled with these bad language habits, which make us sound as though we are hedging and evading—even when we have nothing to hide.
- Does Grammar Count in the Era of E-Mail?** Is e-mail the end of 'correct' communication? Do spelling, punctuation, and grammar matter anymore? Only as much as the recipient of the message matters. All professionals should care about the image they communicate, even in their informal messages.
- Business Basics for Technical Professionals** The most important technical question is "How's Business?" Technical professionals must learn to pitch improvements and changes in their departments through business-savvy business cases: proposals aimed at one's own management. Business cases must show how the new procedures or technology will either make or save money, and within an acceptable number of months.