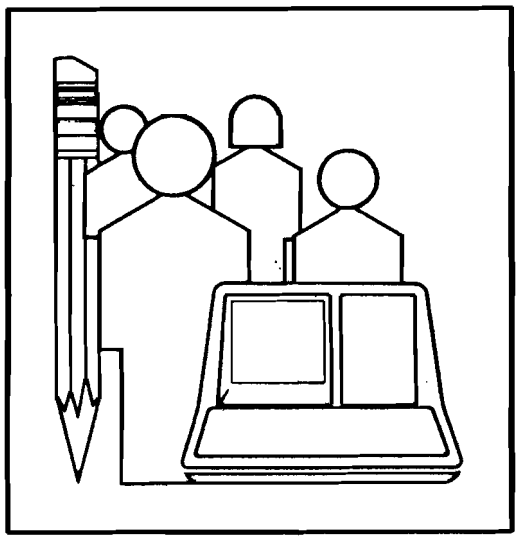

TECHNICAL REPORT

THE PERSONAL COMPUTER AS AN APPLIANCE
PROBLEM I: INTEGRATING TRAINING AND DOCUMENTATION

Technical Report No. 11

August 1981

DIGITAL EQUIPMENT CORPORATION
EDUCATIONAL SERVICES
12 Crosby Drive
Bedford, Massachusetts 01730

digital

This Technical Report may be copied for non-commercial purposes with credit to the authors and Digital Equipment Corporation.

THE PERSONAL COMPUTER AS AN APPLIANCE
PROBLEM I: INTEGRATING TRAINING AND DOCUMENTATION

Jesse M. Heines, Ed.D.

ABSTRACT

Personal computers will become ubiquitous office appliances only when their ease of use is significantly increased by integrating adequate training and documentation with the base system. This integration requires careful attention to human engineering, by providing intuitive access to HELP and on-line training, designing clear and informative error messages, prompting users for input, orienting users to which mode they are in, and presenting information on the screen in such a way as to avoid disrupting the user's job or task context. Such features often require trade-offs between ease of use and system performance. Such trade-offs must be made with a thorough knowledge of the user population and how the system will be used.

This paper has been accepted for presentation at the International Conference on Computers and Society (sponsored by the IEEE) in October, 1981.

THE PROBLEM

The personal computer presents a number of new challenges for those of us involved in training and documentation. It is relatively easy to justify a tuition of $500 for a one-week training course when you have a $100,000 computer, but far more difficult when the computer costs $2,500. Computer costs have plummetted, while training costs have skyrocketed. Yet the need for training still exists, perhaps more than ever. Computer systems are better human-engineered than in the past, but they are also more complex. Software is easier to use, but the people using it are far more computer-naive (and even computer-phobic) than the users most computer professionals are familiar with.

The problem is to provide the needed training and documentation in a manner acceptable to personal computer users and at a price that they are willing to pay. Personal computers will not take their place as standard office appliances until these goals are achieved. Training and documentation must be succinct, but comprehensive. They must be readily accessible, but extensive enough to answer users' questions. They must provide different levels of detail for different users, but not be obtrusive and disrupt the users' work flow. They must be integrated with the computer system so that all components meld into a useful and attractive package.

This paper does not provide formulae for meeting all of the above criteria. Rather, it discusses software techniques that might be employed to help meet the challenges presented and the ramifications of implementing these techniques in a personal computing environment.

CHARACTERISTICS OF APPLIANCES

Appliances are ubiquitous items and have several distinctive characteristics:

- They usually come with very small instruction booklets.
- They have simple, non-threatening appearances.
- They have clearly marked controls whose functions are obvious.
- They have an aura of <u>familiarity</u>.
- Their function and operation can be inferred from their <u>design</u>.

Some would classify the previous characteristics as features that make appliances easy to use. But they do more: they make using appliances intuitive.

Most people learn to use appliances by intuitive trial and error. They take the appliance out of its box, study it to see how its parts differ from other appliances with which they are familiar, make a few assumptions, plug it in, and give it a try. If it doesn't work, they revise their assumptions and try again. If it still doesn't work, perhaps then they turn to the instruction booklet, but they will probably stop reading as soon as they find out what they forgot to do. Their assumptions are based on intuition fed by their past experience and their concept of what this appliance is supposed to do.

Computer systems are not quite so intuitive, perhaps because there wasn't one sitting on the kitchen table when most of us were a kids. Far fewer people have realistic concepts of what computers are supposed to do: if you ask several people how a computer does word processing, you will get answers spanning an extremely wide range.

Computer systems must capture intuitive qualities if they are truly to become as simple to use and thus as ubiquitous as common appliances. The best way to do this is to put more emphasis on human factors in original hardware and software designs, and there is no direct substitute for this effort. But systems offering similar functions will continue to differ greatly due to attempts to optimize specific base hardware and software features. Given these differences and the general lack of prior computer familiarity among most personal computer users, the most viable way to achieve intuitive quality in the immediate future is by integrating the training and documentation with the system itself.

CHARACTERISTICS OF INTEGRATED TRAINING AND DOCUMENTATION

Accessing HELP

Most systems have some sort of HELP function. In many cases, the user simply types the word HELP followed by a system keyword. For example, if a user types:

```
$ HELP DIRECTORY
```

on VAX/VMS, the system responds with:

```
DIRECTORY
Provides a list of files or information
about a file or group of files.

Format:  DIRECTORY [file-spec[,...]]

Additional information available:
/BEFORE[=time]          /BRIEF (D)
/COLUMNS=n (D=4)        /CREATED (D)
/DATE[=option]          /NODATE (D)
     .                       .
     .                       .
     .                       .
```

HELP on this system is arranged hierarchically by command. Information is provided at different levels, and users can get information on the command qualifiers by typing the qualifier as part of the HELP command. For example,

```
$ HELP DIRECTORY/PROTECTION
```

results in the following response:

```
DIRECTORY

/PROTECTION
/NOPROTECTION (D)
 Controls whether the file protection
 for each file is listed.  The default
 is /NOPROTECTION, which does not list
 the file protection.
```

There are two basic problems with this approach. First, users must know what they are looking for before they can look for it. That is, you must know that you want information on the DIRECTORY command to be able to know to type HELP DIRECTORY. If you type:

```
$ HELP FILES
```

because you intuitively think you want information on the files you have stored, the system will respond with:

```
    Sorry, no documentation on  FILES

    Additional information available:
ALLOCATE        ANALYZE         APPEND
ASSIGN          BACKUP          BASIC
BLISS           CANCEL          CLOSE
COBOL           CONTINUE        COPY
CORAL           CREATE          DEALLOCATE
DEASSIGN        DEBUG           DECK
DEFINE          DELETE          DEPOSIT
DIFFERENCES     DIRECTORY       DISMOUNT
DUMP            EDIT            EOD
EOJ             ERRORS          EXAMINE
EXIT            FMS             FORTRAN
GOTO            HELP            IF
INITIALIZE      INQUIRE         JOB
LEXICAL         LIBRARY         LINK
LOGIN           LOGOUT          MACRO
MAIL            MCR             MERGE
MESSAGE         MOUNT           ON
OPEN            PASCAL          PASSWORD
PATCH           PHONE           PLI
PRINT           PROCEDURE       PURGE
READ            RENAME          REPLY
REQUEST         RMS             RUN
RUNOFF          SET             SHOW
SORT            SPECIFY         START
STOP            SUBMIT          SYMBOLS
SYNCHRONIZE     SYSTEM          TECO
TYPE            UNLOCK          VTEDIT
WAIT            WRITE
```

This is quite a formidable list. It does not function well as a menu because many of the entries have relatively little meaning even to average users. (Average users are generally familiar with the subset of commands that they use regularly and may know nothing about those commands that are used very infrequently.) To naive users, a menu such as this can be frustrating and even frightening. More importantly, it provides no clues as to which commands deal with FILES and therefore does little to help the user find the specific information he or she desires.

One approach to providing HELP that addresses these issues is to present users with a menu arranged hierarchically by subject area rather than by command. Even though every computer system seems to have different commands for similar functions, there are

common areas that are intuitive to users with at least some experience. Such areas include:

- accessing the system
- editing files
- storing files
- displaying files
- controlling devices
- controlling processes (jobs)
- compiling (and linking) programs
- running and debugging programs
- communication facilities

If naive users were presented with a task-oriented menu of this type rather than a menu of unrelated commands, they could more easily ascertain how to access the information they are looking for. In the example above, the user who typed:

```
$ HELP FILES
```

when he or she wanted DIRECTORY information would select the option on storing files from this menu and might then be provided with the following submenu:

```
STORING FILES

Additional information available:
CREATE          DELETE      DIRECTORY
EDIT            LIBRARY     PURGE
RENAME          RMS         SORT
```

This is considerably more manageable. Note that some commands, such as CREATE and EDIT, might appear in the submenu for "editing files" as well. There is no intrinsic problem with this, as long as the resulting menu remains small enough so that the user can find the appropriate command simply by scanning the menu and using a bit of intuition. In the example above, inexperienced users might confuse the LIBRARY command (which maintains libraries of object code) with the DIRECTORY command (which lists information on disk files), but it is unlikely that they would choose commands such as DELETE or RENAME. Therefore, the choice is cut down from a menu so large that it provides more confusion than help to one that leaves users with only one or two intuitive choices.

The second problem with standard HELP commands is that users must be at the system's monitor level to use them. If one is using a text editor and desires information on the DIRECTORY command, he or she must close the currently open file, execute a HELP command, and then go back into the editor. This is a tedious task and may have to be repeated several times on a CRT terminal, where information continually scrolls off the user's screen.

The UNIX operating system provides a powerful feature that allows programs to spawn other processes (jobs). This feature is extremely useful for on-line training and documentation because it allows the current user process to be suspended, the user to go do something else in a subprocess, and then resume the original process right where he or she left off. Consider the following scenario as an illustration:

> You are editing a text file that you will "mail" to another user explaining how to use a new program. You are writing the directions and forget the exact syntax of an esoteric command option. Without subprocess capabilities, you must file your text on disk, exit the editor program and return to the system's monitor level, run the utility to check its operation, reenter the editor, find the place at which you were entering text, and then continue editing. With subprocess capabilities, you can suspend the editor program, run the program, and then resume the editing process just as if you had two separate terminals. This procedure saves even more time if the editor takes long to load or if you have a number of temporary editing buffers active at the time.

The ability to spawn subprocesses is even more important in computer-assisted instruction (CAI) applications. Consider the following illustration:

> When teaching about system utilities, it is always valuable to have learners do exercises using these utilities. The courseware typically simulates the utility so that the user's input and the system's output can be controlled, and user errors carefully analyzed and dealt with in an instructionally sound manner. The problem with this approach is that programming of the simulator is redundant with programming of the utility, and any changes in the underlying utility might invalidate the simulation. With subprocesses, users can be allowed to use "the real thing", but their input can still be controlled, the system's responses monitored, and error messages intercepted and appropriately explained.

This technique allows on-line training to be more tightly integrated with the base system than straight simulation.

Error Messages

Error messages are one of the most common sources of naive user misunderstanding because they are usually written for sophisticated users. For example, if one types:

```
$ DELETE ONE.TWO;3
```

to try to delete a non-existent disk file called "ONE.TWO;3", VAX/VMS responds:

```
%DELETE-W-FILNOTDEL, error deleting
   DBA3:[HEINES]ONE.TWO;3
-RMS-E-FNF, file not found
```

The first line of the error message tells what error occurred. The header information for this line indicates the facility running at the time the error was generated (the DELETE program), the error message severity (W for "warning"), and an internal identification of the error message (FILNOTDEL for "file not deleted"). The third line tells why the error occurred: an RMS (record management system) error, whose internal error code is FNF (file not found).

This is a lot of extraneous and confusing information for naive users, particularly when the text suffices alone:

```
Error deleting DBA3:[HEINES]ONE.TWO;3
File not found
```

(Fortunately, a single VAX/VMS command allows just the text to be presented, without the accompanying headers.)

This technique brings up the issue of programming for specific target users and tailoring the messages in programs for their level of understanding. One of the popular editors on DEC systems has a two complementary commands: SET NOVICE and SET EXPERT. These commands set the tone of error messages, respectively wordy and long vs. terse and short. Providing both types avoids having to know the expertise of one's user population, but it does not avoid having to know the extremes for which one must provide such messages. That is, one must still ascertain just how wordy messages have to be for naive users and just how terse they can be for experienced users. Limited memory and disk storage on personal computers sometimes prohibit use of this technique.

As an alternative to multiple error messages, the system might refer to the page in the users' manual where an error is discussed in more detail. The on-line message can then be very concise, with the longer explanation provided in writing. This type of integration is difficult to achieve in practice, however, because manuals are not usually fully developed until after the software code is frozen. In addition, any change in the manual's

pagination would necessitate a large number of changes in the system software. One possible solution to this dilemma is to store the page references in a table that is accessed by the error routines and can be updated independently.

The PASCAL system developed by the University of California at San Diego has a beautiful method of helping the user deal with syntax errors that occur during program compilation. This system identifies the error, chains directly to the screen-oriented editor from the compiler, opens the program source file for editing, and positions the cursor directly under the first errant character. With this approach, the documentation (error messages) are integrated with the system in an extremely novel and useful manner.

Prompting and Orientation

One of the most difficult aspects for naive users is to try to understand what an applications program is looking for when it is waiting for user input. Without a general understanding of the application, naive users find it hard to use their intuition to decipher many prompts. To illustrate this problem, consider the following.

> As part of the registration process for our CAI courses, learners are asked to identify themselves by typing a code name. If they have not yet registered, learners are instructed to simply press the RETURN key without typing anything else. The act of deciding whether to press RETURN or enter a code name turned out to be an extremely difficult task simply because learners were not properly prompted. Our first prompt read as follows:
>
> Please identify yourself by typing
> your code name below and then
> pressing the RETURN key. If you
> have not yet registered on this CMI
> system and selected a code name,
> just press the RETURN key without
> typing anything else.
>
> Your code name?
>
> When this prompt was used, both types of mistakes were made: new students tried to type code names and previously registered students pressed the RETURN key. We found, however, that new students made far more errors

than previously registered ones. We therefore emphasized the action required of new students by enclosing this in a box and wording the instruction as follows:

```
+----------------------------------+
| Press RETURN if you have not yet |
| registered for this course.      |
+----------------------------------+
```

Otherwise, identify yourself by typing your code name. Then press RETURN.

Your code name?

The box helped considerably, but we found that the instructions were still too wordy. So we cut down the number of words again:

```
+----------------------------------+
| Press RETURN to enter the course |
| for the first time.              |
+----------------------------------+
```

Otherwise, type your code name:

When this prompt was used, the number of errors dropped to near zero, indicating that the format allowed even new users to intuitively understand what the system wanted them to do.

A complementary problem to knowing what to type is knowing when to type it. Most systems have at least two levels: the system monitor and application programs. For example, one can't use editor commands when sitting in the monitor, nor monitor commands in the editor. This problem is particularly evident when instructing naive users in BASIC programming. It seems to take them a while to realize they must be "in BASIC" before typing program lines, and that they must "exit BASIC" before performing system functions such as copying a file. This confusion occurs even with BASIC printing "Ready" as a user prompt and the monitor using a character prompt such as $, >, or].

These prompting problems relate to the more general problem of system orientation, an issue with small systems as well as large ones. Personal computer users forget which disks they have inserted, which mode they are in (text or graphics), and even

which application they are running (particularly when they may have two or more editors available).

Again, these problems are even more important when one tries to [put training and documentation on-line. With paper documentation, users can orient themselves via page numbers, running heads, and even how far along they are in the manual. That is, users always know how much more they have to read just by noting their place in the manual. This isn't true with on-line training and documentation: if the screen simply says "Press RETURN to continue", users have no way of knowing how many times they'll have to do that before they complete the current section.

Several techniques can be used to improve orientation. The simplest is to identify relevant information in some fixed portion of the screen, such as the upper right-hand corner. This is the equivalent of running heads in paper-based documentation. One might identify the label of the current disk, the screen mode (text vs. graphics), and the name of the current application. The important point is to provide only that information which is relevant to the user, and to keep this information clear enough for naive users to understand. Nothing is quite so frustrating as to ask, "What's that mean?" and to be told "Oh, don't worry about that."

Another technique is to color code titles and other header information to provide visual clues to one's whereabouts. On black and white terminals, one might be different type styles, different screen formats, or some other clues to provide orientation. The important point is simply to give users some indication of where they are so they don't get lost in a HELP sequence trying to figure out that they have to press RETURN before the system will recognize the command being discussed in the help sequence.

A third technique is to change the actual prompt that appears when the system is waiting for input. We have found that single line prompts are superior to ones that appear above the user's input. For example, most BASICs simply print:

```
Ready
```

(followed by a carriage return line feed pair) to identify themselves and indicate that the system is waiting for additional user input. A better approach would be to prompt users like this:

```
BASIC >
```

(without the carriage return line feed pair) and have users type their next command or statement right after the >. This assures that the user realizes that the prompt is a prompt and not part of any previous output.

This "labelled prompt" technique can be very effective with application programs. Instead of just expecting users to know that they should type something, prompt them with the program name:

Accounts Payable >

This assures that users know not only when input is required, but also gives them a hint as to what input is required. The beauty of this technique is that it works well with both naive and experienced users: it is informative yet very concise, and orients the naive user without getting in the experienced user's way.

Windowing

All of the techniques discussed above relate to how users access information and the style in which that information should be presented. Attention to such details can contribute greatly to simplifying personal computers and making them more approachable. But no matter how much human engineering one builds in, computer systems will never be as intuitive as the common appliances because computers are just too complicated to expect that all of their functions can be used effectively by any Tom, Dick, or Harry without adequate training and documentation. HELP will always be needed, and the jobs of today's trainers are quite secure.

Given knowledge of what users need to know, how that information will be accessed, and the style in which it will be presented, the problem becomes one of designing how training and documentation should be displayed on the user's screen.

There are two basic options: one can erase the user's screen before he or she accesses on-line training or documentation, or one can present the training or documentation in a "window" in a defined area of the screen. The first option certainly simplifies programming, but it has a very important disadvantage: if a user makes an error and tries to use on-line training or documentation to understand the source of that error, the error will be erased while the explanation is being given. The second option not only avoids this problem, but is especially useful when combined with subprocesses so that users can, in effect, run a number of virtual terminals from a single terminal. Note that these are interactive virtual terminals, unlike those spawned by submitting jobs to batch processors.

Windowing is not a new technique. It has been used on Smalltalk at Xerox PARC, has been the subject of intense study by the Computer Science Department at Brown University, and is an integral part of Apollo computer systems. In these cases, windowing has basically been used for interactively controlling independent subprocesses. What is proposed here, however, is to use window-

ing to provide users with training and documentation in such a manner as to let them maintain their current job or task context while the requested information is being displayed.

The ideal approach is to make the system services needed to support windowing an integral part of the operating system. This avoids having to program them into every application independently. With this capability, on-line training and documentation automatically become integral parts of the system because they are callable from any application.

CONSIDERATIONS

The techniques discussed in this paper are not without cost ramifications. Some of considerations in employing these techniques are as follows.

- Extensive HELP and CAI facilities require a considerable disk space.
- Subprocesses and windowing require enough memory for a sophisticated operating system.
- Creative error recovery techniques require enough CPU power to provide human-engineering without aversely affecting system performance.
- Clarity of communication requires screen and graphic capabilities to provide effective prompting and orientation.
- Human engineering is a development cost in itself.

All of these techniques require trade-offs. No matter how big a system is, one can always deteriorate its performance by adding a large enough amount of overhead. Just how much one can afford to integrate training and documentation with a system may depend heavily on one's market. Elimination of human-engineering will allow one to manufacture hardware and software cheaper, but the resultant product may not be adequate for specific classes of users.

The problem is not usually in determining how many of these features one can afford to provide, but rather which ones one can't afford not to provide. Inadequate training and documentation will decrease the usability of personal computers and increase the providers' support costs.

This paper indicates some of the techniques that can be used to increase the intuitiveness of computer systems and thereby increase their ease of use. The discussion is neither in-depth nor exhaustive. Yet the basic premise is clear: training and documentation must become integral parts of personal computers if these valuable tools are to become ubiquitous appliances in the office environment.

ACKNOWLEDGEMENTS

Michael Zimmerman of Digital Equipment Corporation was instrumental in developing several of the ideas presented in this paper, particularly those relating to subprocesses. Roger Bowker of Digital increased the author's sensitivity to the power of wording and text layout to increase the efficiency of communication. Victor Bunderson of WICAT, Inc. and Wendy Mackay of Digital stimulated the thinking on orientation. Andries van Dam, Steve Feiner, and Norm Meyerowitz of Brown University contributed heavily to the ideas on windowing.