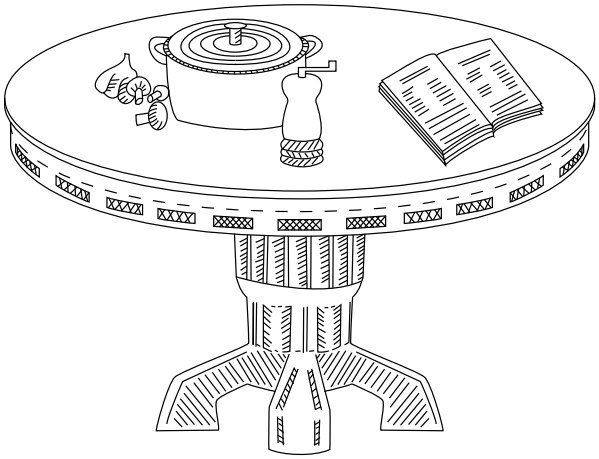
HUMAN FACTORS LESSONS
from the Design of a Real Product

Abstract

Good human factors was a major design objective for the Tektronix 4991S1 Graphic Input Workstation. For the most part, this objective was achieved. Nevertheless, this system has some human factors deficiencies. This paper analyzes the design and implementation process, and relates the process to the strengths and weaknesses of the result. The paper concludes with a summary of the human factors lessons we learned, our mistakes and their consequences, what we would do differently and what we would do the same.

Herb Weiner
Tektronix
P.O. Box 1000
Wilsonville, Oregon 97070
(503) 685-3586

Tektronix®
COMMITTED TO EXCELLENCE



Kitchen Wisdom Publishing

Gourmet Software
10032 SE Linwood Avenue
Portland, Oregon 97222
(503) 771-1402
fax: 771-1401
http://www.wiskit.com

Retrospective

First, I'd like to thank Tektronix for allowing me to republish this Technical Report. An earlier version of this report was first presented in February 1985 at the SHARE Conference in Los Angeles. It was revised and released as this Technical Report a few months later. Although I've considered numerous improvements to this Report over the years, I ultimately decided to republish the Report in it's original form.

Like the Xerox Star and the Apple Lisa, the Tektronix 4991S1 Graphic Input Workstation is only a memory. I'd like to think, however, that the influence of the 4991S1 lives on. This product was introduced around the same time that Apple introduced the Mac Plus. It featured an E-Size (34 by 44 inches) 300 DPI scanner (the Autovectorizer)that produced vectorized (centerline and outline)output in real time. (Even at 1 bit per pixel, producing a raster image would have required over 16 MB of memory, and would have been unthinkable at the time.)

The Graphic Structuring Software pioneered the user interface concepts later adopted by Adobe Illustrator and other software, in particular, the use of the scanned input as a template which could be traced by the user. When did Adobe first obtain a copy of this Report, and how much did it contribute to the user interface of Adobe Illustrator? Only Adobe knows for sure. However, in May, 1986, I did personally give a copy of the Report to Doug Brotz, a Senior Computer Scientist at Adobe. And the first verrsion of Adobe Illustrator was not released until early 1987.

Desktop Publishing did not yet exist when I started work on this paper. Typesetting was handled by a separate department. Thus, when I decided to republish this paper on the World Wide Web, I had no PostScript to feed to the Distiller, and was forced to reenter both the text and graphics. As much as possible, I've tried to preserve the appearance of the original report. Thank you, Adobe, for Illustrator 5.5 and PageMaker 6.0, which helped facilitate this effort.

Herb Weiner (herbw@wiskit.com)
September, 1995

Introduction

In his book *Travels in Computerland*, the author, Ben Ross Schneider Jr., describes the painful experience of "converting *The London Stage*, an 8000-page calendar of performances from 1660 to 1800, to a computer accessible information base." Although he had hoped to use optical scanning techniques to enter the information, it was ultimately necessary to key the information in manually. Imagine how much more difficult the problem becomes when the information to be converted includes graphics as well as text!

Tek Labs began investigating the application of scanning technology to input graphic information in 1978. A working prototype which later became known as the autovectorizer was developed. It was a raster-based optical scanner that could scan an 8½ x 11 inch drawing and produce vector-format output in real time. This technology was transferred to the Information Display Division in 1980. At that point, Tektronix did not know exactly how to use the technology; it was a solution looking for an application.

It took four more years for our group to incorporate the Autovectorizer technology into a complete system (the 4991S1 Workstation), making it practical to enter existing drawings into graphical databases.

Developing the Autovectorizer prototype into a high-quality product that could scan an E-size (34 x 44 inches) drawing was no easy task. But developing the Autovectorizer into a product was only part of the problem. To make the output useful, we had to find a way to add semantic information (structure) to the output of the autovectorizer. For example, mechanical drawings contain dimension information (arrows and text) which humans interpret differently from other information in the drawing. Simply placing the vectors forming the dimension arrows and text into the customer's database as if they were part of the geometry of the drawing would not produce a useful result.

Initial Direction

Initially, we assumed that the structuring process would involve editing the Autovectorizer output. After pursuing this approach full time for several months, we began to realize that even with a special purpose editor, the editing approach would be tedious because the customer would need to replace (delete and insert) a large percentage of the drawing. Editing is suitable for making minor revisions, not for restructuring an entire drawing! Furthermore, we were trying to design an input system only. We assumed that customers would subsequently use existing CAD tools (including editors) to make modifications. Since we wanted to produce a system which could be used with a variety of CAD systems, any editor we could design for structuring would be inconsistent with and would duplicate some of the functions of the editor the customer was already using. We had no intention of completely replacing existing editors, so the customer would have been required to use two inconsistent editors. Finally, the editing model did not support the types of interactions required; for example, entering text so that it lined up (scaling, rotation, position) with the vectors being replaced. If we had pursued the editing approach, we would not have produced a winner.

Instead of assuming initially that the structuring process would be based upon a special purpose editor, we should have begun by analyzing the requirements.

Because we considered the task of designing a special purpose editor to be a straightforward process, assigning an engineer to work on the editor was delayed until the Autovectorizer project was well underway. This mistake wasted precious time. The engineering resources were finally allocated to the project, but it was several months before we began working on the real problem: designing structuring software (rather than editing software).

Requirements Analysis

There is a lesson here. The human factors of many products suffer because the design is based upon a poorly chosen (wrong) set of requirements. The author believes that design methodologies that advocate producing a "Requirements Specification" document conforming to *IEEE Guide for Software Requirements Specifications*, or similar standards, are particularly susceptible to producing poorly chosen requirements when applied to products with complex user interfaces. The problem is that such Requirement Specification documents are attempts to place both requirements and the description of a specific implementation (not always the best one) into a single document.

The author believes that better products would result if two separate documents were produced: First, a "Requirements Analysis" document, which would analyze the costs and benefits of potential product features and performance. After all parties approve the Requirements Analysis, it is appropriate to begin work on a "Product Specification" document describing a specific implementation including those features from the Requirements Analysis document which best meet the cost, schedule, and performance goals.

The need to prototype software is widely recognized. In *The Elements of Friendly Software Design*, Paul Heckel notes (page 130) that "If you accept the premise that designing friendly software is a communications craft, and we can learn from the experiences of other craftsmen, you must accept the necessity of constantly revising your programs to get them to be easy to use. We cannot expect success the first or second time. If our program is at all innovative, we can't expect it to be right even on the third or fourth time."

The Product Specification should be a living document that is updated as the prototyping progresses. The Requirements Analysis document can help to provide direction to the prototyping activity.

The basic requirement analyzed in our Requirements Analysis document was that we be able to enter existing drawings into a variety of CAD systems with improved speed and accuracy compared with existing approaches. Using an editor to accomplish this task was **not** a requirement (although it could have been a solution). We studied three competing techniques: manual digitization, design on the screen (completely reconstructing an existing drawing), and automatic or semiautomatic scanning. As we analyzed requirements for different applications, we realized that the requirements for positional accuracy were different for different applications. In fact, drawings contain two kinds of information: spatial, or positional, and symbolic. Maps generally contain much more spatial information than symbolic information. Mechanical drawings tend to be more balanced, while electrical schematics contain primarily symbolic information. Manual digitizing is generally used only in applications where the spatial information is significant, and cannot readily be supplied using alternative techniques. It is a slow, tedious, and imprecise process. Design on the screen is generally used when the spatial information is not significant (for example, electrical schematics), or where it can be entered numerically (for example, from the dimension lines on a mechanical drawing). Automatic scanning overcomes the disadvantages of manual digitization: it is fast and accurate. However, automatic recognition of symbolic information has not yet advanced to the point where automatic scanning can compete with design on the screen for applications where drawings contain a high ratio of symbolic to spatial information. This analysis helped us to narrow our focus to applications in which drawings contained a significant amount of spatial information.

Initial Conceptual Model

The first model developed was based upon manual digitization. We would make the workstation emulate an "intelligent" graphic input (digitizing) tablet. Imagine that we could give a manual digitizing operator a custom plastic template containing all the features of the original drawing. The operator could rapidly and accurately trace (and enter into the CAD system) all or part of the original drawing. To illustrate this model, Figure 1 shows the custom plastic drafting templates produced when the Autovectorizer processes a tax map, a mechanical drawing, and a circuit board layout. The operator could select a property boundary, a portion of the mechanical part, or a circuit board run, and our "intelligent" tablet could rapidly and accurately "trace" the selected portion of the template and transmit the coordinates of the points along the lines to the CAD system.

A manual digitizing operator does not trace all the lines on a drawing to be digitized. Text is entered using the keyboard. Our "intelligent" tablet also requires that the operator key in the text, assuming that the text must be stored in the CAD system as characters rather than vectors. For example, the operator would probably key in the numbers representing the lot numbers of the property parcels. Our "intelligent" tablet would automatically determine the position, size, and rotation angle for the text based on the vectors forming the text in the original drawing.

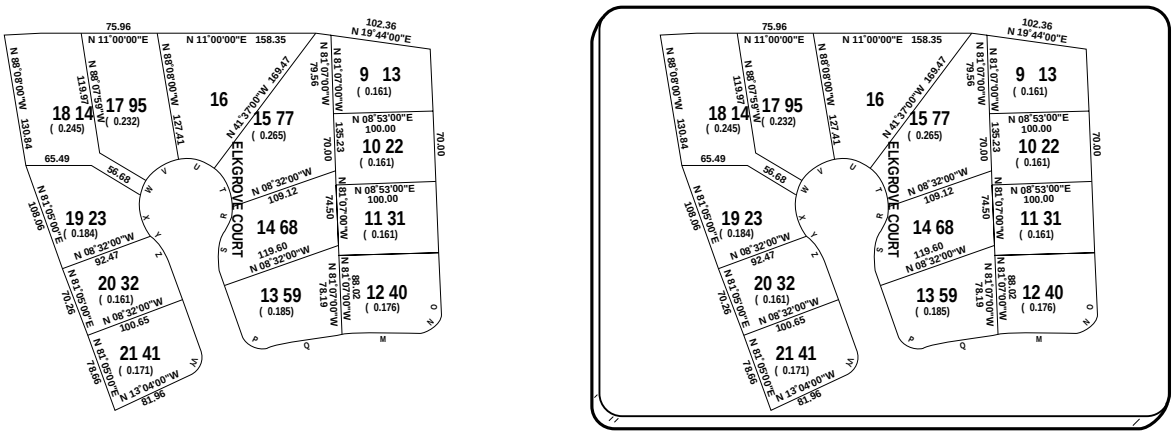
Although our model was based upon manual digitization, it would result in higher accuracy and higher productivity and be less tedious than manual digitization. Rather than dumping vast amounts of unstructured information (garbage) into the database to be corrected later (using an editor), we would place the operator in control, giving him the freedom to trace, using a graphic tablet, only those features of the template that he would trace if manual digitization were being used.

This model was based upon creation (input) rather than editing (revising), and was a key breakthrough! This approach involved another new concept. With most editors, the operator can move around the document and revise sections, even if they have already been edited. The accumulated revisions are saved all at once, when the operator SAVEs the document or exits from the editing session. In contrast, manual digitizing is generally a sequential process in which portions of the drawing are saved as they are entered by the operator. Rather than sending the entire drawing to the CAD system only after the editing was completed (an approach that was dubbed SPLAT), we would be sending the drawing continuously during the structuring session (an approach that was dubbed DRIBBLE).

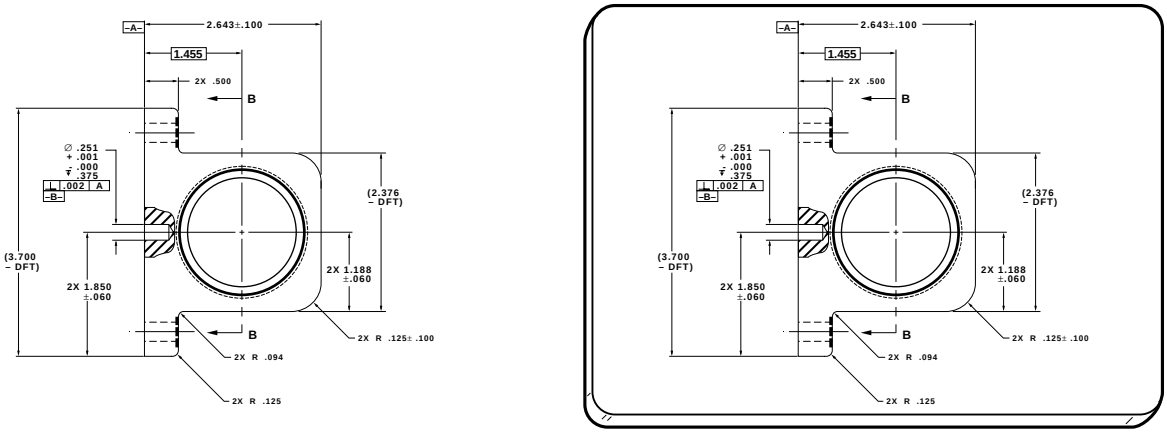
We divided the tablet into three regions. One region would be used for graphical interaction with our workstation. A second region would be used for command interaction with our workstation (through a menu). A third region was reserved for interaction with the CAD system and would be passed to the host without any local processing.

We decided to use three colors on the screen for the display of the drawing. The first color would represent the template. The second would indicate buffered information which had not yet been transmitted to the CAD system, while the third would indicate information which had been transmitted. We would not allow any modification of information once it had been sent to the CAD system, except by using the CAD system editor upon the completion of the structuring process. We would also allow portions of the template to be temporarily hidden to reduce clutter.

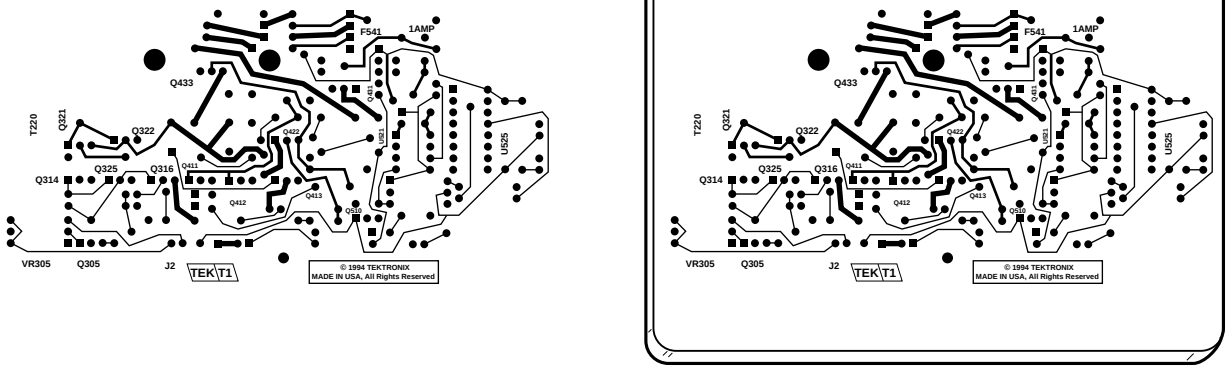
ORIGINAL DRAWINGS **CUSTOM TEMPLATES**



MAPPING



MECHANICAL



ELECTRICAL

Figure 1. The TEKMAN conceptual model. The Autovectorizer produces custom plastic drafting templates from drawings.

First Prototype

The first prototype looked nothing like the final product. We had been looking for ways to make the process of entering data less tedious, and began talking about what makes video games fun. Some of the characteristics we identified were challenge (requires skill), interactivity, and control. Inspired by a popular video game, we wrote TEKMAN[1], the TEKtronix MANual digitizing system. There was an element of skill involved in steering the cursor along features of the template. Although this "game" was fun to play (we had even jokingly mounted a coin box on the system), it provided a powerful demonstration that we could achieve significant gains in both speed and accuracy compared to manual digitization.

It was fortunate that we completed the first prototype when we did. At this time the U.S. economy had become sluggish, and Tektronix was looking for ways to reduce expenses. As a result, our project was cancelled! Actually, it is rather common in high technology industry to invest in high risk projects, and cancel those that fail to show promise (survival of the fittest). However, those high risk projects that do work out often yield big payoffs.

Thanks to an extremely enthusiastic manager, our engineering team was granted a two week reprieve. During that time, we demonstrated our prototype to many skeptics. The general comment was that they had had no idea the product was capable of such performance. Our temporary funding was extended several times, until permanent funding was finally restored.

There were several problems with the first prototype. The CAD systems with which we hoped to interface were designed with a human operator in mind. Such systems might not be able to receive data at the rate we would be able to send it. This was especially true of systems which generated vast amounts of feedback (intended for the human operator), which incidentally, we would not be able to use (remember, the screen would be used to display our template). But the biggest surprise to us in engineering was that the marketing group objected to this model. They explained that potential customers for the Workstation might be reluctant to investigate a product modeled after something as unpopular as manual digitizing.

We were therefore left with a user interface, but no conceptual model. From a human factors standpoint, we were unhappy with this situation, since it is generally easier for users to learn a new system if they are presented with a good conceptual model. Heckle (page 94) points out that "The reality of what our programs actually do is of no importance to the user; only the mental model of what they do is important. Our task as designers is not so much to design and create software that carries out a function as it is to design and implement a believable and communicable illusion, or model, for our users."

We realized that to overcome the limitations of CAD systems designed for use by human operators, we would need to provide a host-based interface program for each CAD system. This program (which came to be known as TekCouple[2]) would be designed to accept incoming data at a higher rate than the interface written for the human operator, and would not attempt to send graphical feedback to TEKMAN (TEKMAN provided its own graphical feedback). If TekCouple detected an error (e.g. referencing an undefined symbol), it would send a message to TEKMAN to be displayed for the operator. Because TEKMAN would be communicating with a cooperating program (rather than a vendor program over which we had no control), we would have more control over the design of the user interface (and user model).

[1] "TEKMAN" is a trademark of Tektronix, Inc.

[2] "TekCouple" is a trademark of Tektronix, Inc.

An Improved Conceptual Model

System development proceeded without a formal conceptual model for several months before we finally developed a model that was acceptable. A draftsman works with a variety of tools to create drawings on paper: pencil, eraser, pen and ink, various templates, straightedge, compass, etc. The Autovectorizer would provide the custom template; the remaining tools used by a draftsman would be provided through the user interface. What we previously called the buffer became the penciled portion of the drawing. The operator could erase penciled features to correct errors. When the operator was satisfied with the penciled features, they could be inked. At this point, the information would be sent to TekCouple, and could no longer be changed (except by using the editor supplied with the CAD system).

We were providing the user with a "toolbox," from which the appropriate tools for each job could be picked (e.g. straightedge, compass, etc.). The choice of tools, and the order in which they are used, is a matter of personal preference. Note that such a toolbox approach is only feasible if each tool operates on a well-defined internal data structure.

Construction tools are one variety of tools in the toolbox. The straightedge is used to construct straight lines, while the compass is used to construct arcs or circles. An arrow construction tool forms straight or curved arrows. Perhaps the most interesting construction tool is a continuously variable ellipse template, which can readily be adjusted to fit any ellipse. Each of these construction tools provides for an optional "cleanup" step which removes the vectors being overlaid by the new feature. These construction tools are extremely important in mechanical engineering applications.

Selection tools are used to select, that is, trace selected portions of the custom template. Unlike construction tools, selection tools never create new features. A variety of selection tools provides the operator with the ability to select individual vectors or entire groups of vectors. For example, the polygon cursor is used to select polygons, and is oriented toward mapping applications. Selection tools can be used to erase, as well as to pencil.

Replacement tools are used to replace selected features with symbols or text. These tools provide automatic position and size calculation (with automatic rotation calculation for text only), with manual override provided graphically (using the tablet) as well as numerically.

Template manipulation tools are used to optimize the custom template for the application. For example, the N-Point Alignment tool uses a best-fit algorithm to position, rotate, and scale the template to fit any application requirements. The Snap-to-Grid tool can be used to force all or selected points onto a rectangular grid.

We finally had a good conceptual model of the system. However, the engineering team retained the TEKMAN name as it seemed to capture the *esprit de corps* of the team members.

Because the TEKMAN project had gotten off to a late start, we were placed in a constant catch-up mode (compared to the Autovectorizer). This would have an impact for the remainder of the project (manuals, evaluation, marketing, as well as engineering). We really didn't recognize until this stage in the project that the complexity of the user interface would make the structuring software as complex as the scanning and vectorizing software.

Interfacing with Other Systems

We were unable to provide a good mechanism to handle messages from the system or from another user. Such messages are normally intended for a human reader, but all messages sent by the host are intercepted by TEKMAN. Unsolicited messages appear to be data communication errors, and are ignored. Note, however, that messages generated by TekCouple are transmitted in the proper packet format and are recognized by TEKMAN and displayed for the operator.

TekCouple is written in FORTRAN, and consists of a host-independent portion and a host-specific portion. This architecture simplifies the task of developing interfaces to additional CAD systems. Furthermore, since Tektronix distributes TekCouple in source form, it is rather straightforward for customers to modify TekCouple to support additional entities. This human factors consideration is important to those responsible for installing and maintaining the system.

Evaluation Phase

Because of the "toolbox" approach, we were able to make periodic releases of the system to an evaluation group. This group was then able to test some of the tools even though others had not yet been completed. This approach allowed us to reduce the elapsed time between the completion of the implementation process and the completion of the evaluation process. Further, it provided more rapid feedback of problems to engineering.

To obtain as much information from evaluation as possible, we did not impose restrictions on the type of problems that could be reported. In particular, human factors problems were reported in the same manner as other types of bugs. All problems received were evaluated and assigned a priority, so that they could be dealt with based upon their priority, rather than based upon whether or not the operation was according to specifications.

One of the considerations in establishing priority was the impact of delaying the fix. Sometimes, it can be difficult to correct a human factors problem after the product is released, because users become dependent upon the way the system operates and will complain (with some justification) that the "new, improved" version is incompatible with the previous version. Other problems may not fall into this category.

The Pressure of a Schedule

A high-technology product can only succeed if it makes it to market in a timely fashion. Thus, the development effort involved constant schedule pressure. Many questions arose while we were designing the Workstation. A variety of techniques were utilized in our quest for answers. Sometimes, we were able to rely on design guidelines or the assistance of experts for answers. At other times, prototyping and/or informal surveys provided some insight. However, we simply did not have time to research every question. Sometimes, we were forced to base our design on a best guess, but whenever possible we made the design flexible.

Unfortunately, we were unable to utilize user interface design guidelines as much as we would have liked. The best set of design guidelines we have found is that developed by Smith and Aucella. Although these guidelines provided helpful advice on a wide range of interface questions, no guidelines were available on graphic data entry, our primary concern.

Several times we were faced with a serious dilemna. We did not understand a particular function well enough to complete the specification, and had no time to build a prototype. We were forced to prototype anyway! When the prototype was finished, it was documented and became a part of the product. This approach was bad because the Product Specification was not completed until the implementation was nearly completed, making it difficult for the manuals group to produce user manuals and for the evaluation group to form a test plan.

Strengths

The system can be used to increase both the speed and the accuracy of entering existing drawings into a variety of CAD system databases, and it is still fun to use. This is really the bottom line!

The toolbox approach we took resulted in a highly modular design. It required a well-defined data structure and well-defined interfaces between the modules. This made it easier to stage the implementation, decreasing the impact of designing and implementing before the design was completed. Having something working (even with limited functionality) made subsequent specification, design, implementation, and testing easier. This approach also improved the reliability of the final result (there are fewer bugs than there would have been with a monolithic design).

A flexible system allows each user or customer to configure the system to conform to individual preferences. This is especially true for products that will be used with systems provided by a variety of vendors. Our Workstation is highly configurable. For example, TEKMAN requests confirmation before performing potentially destructive operations (such as a global erase). However, an experienced operator can turn off the need for such confirmation selectively or globally.

We tried to make the Workstation as configurable as possible. For example, we have not been able to determine the best colors to use for the background, the template, and the penciled and inked features. (Maybe there is no **best** set of colors.) Experts tell us that we should use desaturated colors to reduce eye fatigue. However, it is hard to find four desaturated colors that are pleasant to look at (do not clash) and that are sufficiently distinguishable from each other (even for colorblind operators). We have provided a reasonably good (although not perfect) set of defaults, and have provided a mechanism for each installation or user to override those defaults.

Weaknesses

Some human factors deficiencies resulted from making trade-off decisions between computational or memory requirements and human factors. For example, we decided not to retain certain information once it had been transmitted to and acknowledged by TekCouple. Because this information is not retained, we are able to input more complex drawings (a marketing requirement). For example, the operator can tag a line with a line style attribute (e.g. light, medium, or heavy, solid or dashed) to be used by the CAD system in displaying the line. Once the line is transmitted to and acknowledged by TekCouple, the only way for the operator to determine or change the line style associated with the line is to exit from TEKMAN and use the CAD system. Note, however, that TEKMAN does use a different color to display features which have been inked.

When two lines cross, this fact is normally recorded in the data structure. However, if the user creates new lines (not present in the original drawing) using the straightedge tool, we do not search for and record all intersections with existing lines. Normally this is not a problem; however, confusing situations can arise. For example, the operator can create a polygon which TEKMAN will not be able to find, due to the unrecorded intersections. We felt that the benefit of recording these intersections was not worth the computational complexity involved.

Some human factors problems were imposed by limitations of the base system. TEKMAN runs under Local Programmability (CP/M) on a Tektronix 4115B Computer Display terminal. The 4115B is really an excellent terminal, but it does have some deficiencies that we could do nothing about. For example, certain terminal status (e.g. the visibility of the dialogue area) can be controlled by the user from the keyboard, but can only be tested from within a program by requesting and parsing a character string "report". We simply could not monitor this status on a regular basis and still maintain the highly interactive response required. As a result, compromises were made in the user interface design. Similar problems resulted from limitations in the firmware for the graphic tablet and the color hard copy unit.

Because of the distributed architecture of the Workstation (TEKMAN runs under Local Programmability on the 4115B and communicates with TekCouple, which runs on the CAD system), some human factors problems are inevitable. Some errors, such as referencing a symbol which is not defined on the CAD system, cannot be detected until the information is transmitted. Thus, there is a potential delay between an operator action which causes an error and the time that it is reported. Note that the operator is given an opportunity to correct such errors, so they are not fatal.

The Workstation works as well as it does because we chose an appropriate division of labor between human and machine. As McCormick (page 461) points out, humans are generally better than machines in their ability to "Recognize patterns of complex stimuli which may vary from situation to situation, such as objects in aerial photographs and speech sounds." We therefore decided to make all recognition tasks the responsibility of the human, and we do not support text or symbol recognition. Thus, the operator must key in all text which is to become part of the drawing being entered. Although it is not clear whether text recognition would be fast enough or accurate enough, this has been one of the most requested features among those who have seen demonstrations of the Workstation.

Finally, some human factors problems were deliberately not addressed because of schedule pressure. Human factors, like all product features, have a specific market value. It is a mistake to either underrate or overrate that value. If the value is overrated, it may become necessary to sacrifice other, more important product features. We recently had the opportunity to demonstrate the Workstation to Tom Gilb, a prominant human factors consultant. He asked a rather embarrassing question: How come we had not used icons on our menu to make it easier to locate the desired function? With the pressure of the schedule, it had simply never been considered.

Results and Lessons

Designers should be careful to separate the process of requirements analysis from the process of product specification. Don't let yourself be misled by descriptions of an implementation that are mislabeled as specifications of requirements. We almost made this mistake (designing an editor), which would have been fatal (the project would have been cancelled).

Designers should develop a sound conceptual model of the system. Such a model will provide a sound basis for user interface design decisions and for limiting the scope of the project to something which can be completed. In our case, we recognized that we were building an input tool rather than an editing tool, and thus did not provide traditional editing functions.

In *The Mythical Man-Month* (page 116), Brooks points out that "The management question, therefore, is not *whether* to build a pilot system and throw it away. You *will* do that. The only question is whether to plan in advance to build a throwaway, or to promise to deliver the throwaway to customers. Seen this way, the answer is much clearer."

Time should be allowed for prototype development and revision, especially where a complex user interface is involved. Avoid the trap of postponing the specification until the implementation is nearly complete. We would have had more time for prototype development and revision and been able to complete the Product Specification earlier in the development cycle, had we not be caught in the trap of misstated requirements.

An evaluation group can help engineers design a better product. Don't lose valuable information by making it difficult for them to report human factors problems. Begin the evaluation process while there is still time to make changes. Don't restrict the evaluation group to evaluating only for conformity to specifications. Listen to their human factors complaints; if you don't, you will hear the same complaints from your customers.

Bibliography

Brooks, Frederick P., *The Mythical Man-Month*, Massachusetts: Addison-Wesley, 1975.

Heckel, Paul, *The Elements of Friendly Software Design*, New York: Warner, 1984.

The Institute for Electrical and Electronic Engineers, Inc., *IEEE STD 830-1984, IEEE Guide for Software Requirements Specifications*, New York: IEEE, Inc, 1984.

McCormick, Ernest J., *Human Factors in Engineering and Design*, New York: McGraw-Hill, 1976.

Schneider, Ben Ross Jr., *Travels in Computerland or, Incompatibilities and Interfaces*, Massachusetts: Addison-Wesley, 1974. (Out of Print. Library of Congress Catalog Card No. 74-9250. ISBN 0-201-06737-4)

Smith, S. L., and Aucella, A. F., *Design Guidelines for the User Interface to Computer-Based Information Systems*, Massachusetts: The Mitre Corporation, 1983.

About the Author

Herb Weiner is a Senior Software Engineer in the Information Display Group at Tektronix, Inc. Before coming to Tektronix in 1978, he was a Senior Systems Programmer at Cornell University. He is a member of ACM SIGCHI, SIGGRAPH, and SIGDOC, the Computer-Systems Group of the Human Factors Society, and IFIP Working Group 6.3 on Human-Computer Interaction. Herb was an invited participant at the 1983 Vail Workshop on Human Factors in Computer Systems, where he served as Chairman of one of the five working groups. He received his BS degree in Engineering from Cornell University in 1971.