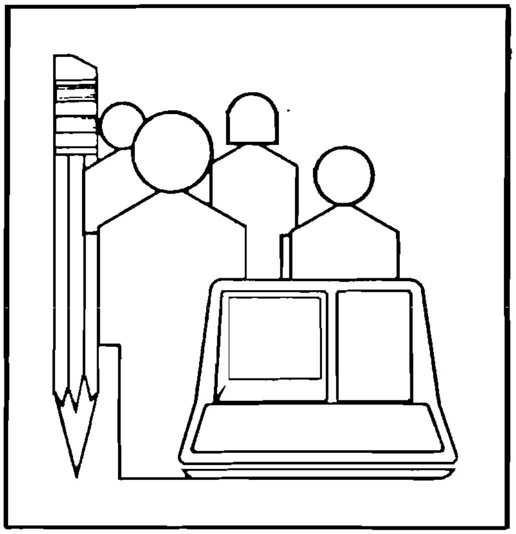
TECHNICAL REPORT



STYLE AND COMMUNICATION IN INTERACTIVE PROGRAMMING

Technical Report No. 3

June 1979

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.

STYLE AND COMMUNICATION IN INTERACTIVE PROGRAMMING

Jesse M. Heines, Ed.D.

ABSTRACT

Research on man-machine communication was examined to gain insight into techniques for improving interactive programs through the enhancement of communicative style. The human-computer interaction is compared to a conversation, and specific recommendations for improving this interaction are enumerated. The suggestions are general in nature and are arranged into a preliminary "Guide to Style in Interactive Programming".

An earlier version of this paper was presented at the 1975 Summer Conference of the Association for the Development of Computer-based Instructional Systems, Portland, Maine, August 1975. It is available from ERIC as Document No. ED111420.

INTRODUCTION

Few people who have interacted personally with a computer have lukewarm feelings about the experience. Most often, people find the interaction either highly enjoyable or totally distasteful (Martin, 1973; Melnyk, 1972). The strengths of these reactions are an important consideration in the design of computer-assisted instructional (CAI) programs, as students who find the interaction distasteful will be reluctant to use the computer repeatedly.

It is this author's belief that most of the distasteful qualities of interactive computer programs may be attributed directly to their poor communicative style. This is because program authors often fail to consider the posture of the naive user. This paper attempts to provide guidelines for good communicative style by examining the research on man-machine communication and techniques that can be used to smooth the human-computer interface.

COMPUTER INTERACTION AS A CONVERSATION

Nickerson (1969) suggests that little work in the related fields of ergonomics, human factors, and human engineering can be applied directly to the design of effective human-computer interactions. "What makes the man-computer interaction qualitatively different from other types of man-machine interactions", he explains, "is the fact that [man-computer interaction] may be described, without gross misuse of words, as a conversation". This theory can be supported by comparing Schramm's model of communication (1954), shown in Figure 1, to a diagram of the processes involved in human-computer interactions, shown in Figure 2.

From these diagrams, it might appear that the ideal human-computer interaction would be an exact replica of a human-human conversation. Chapinis (1971) and Foley (1973) point out several reasons why this is not yet technologically possible, and even a brief examination of this approach will show why it is not desirable in many applications. For example, graphic display techniques can impart far more information than verbal channels (Martin, 1973), and interactions with computers can improve upon normal technical conversations by reducing redundancy (Nickerson, 1969). The most desirable type of human-computer <u>interaction</u>, then, might be described as the one which allows the most efficient operation of the human-computer <u>system</u>. That is, it should be designed to provide the easiest-to-use interface between the

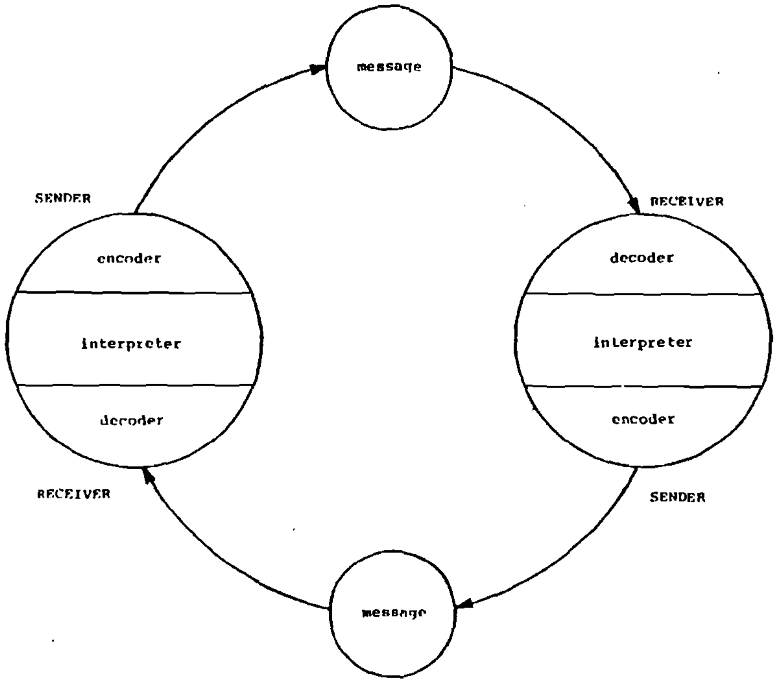


Figure 1

Schramm's Model of Communication

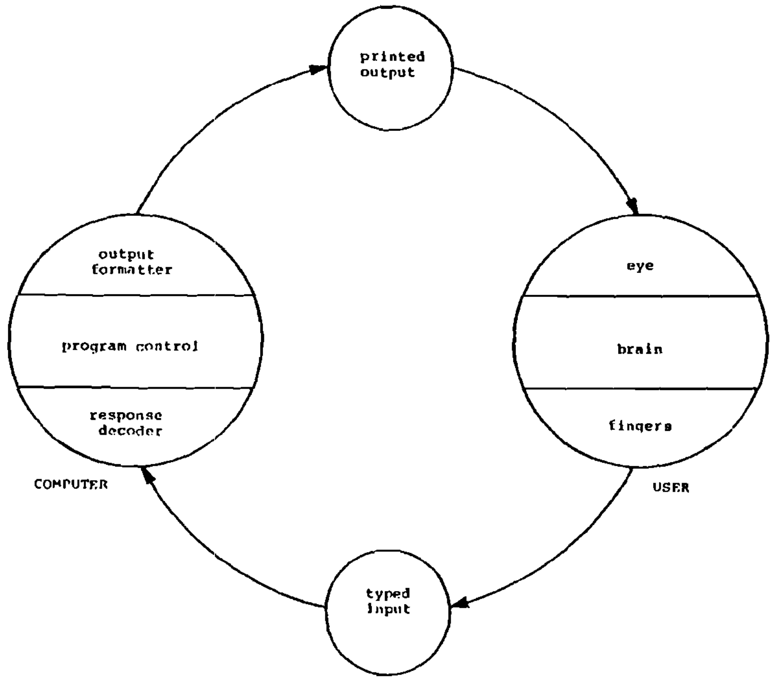


Figure 2

Human/Computer Interaction

problem defined by the user and the related capabilities of the computer (Foley, 1973; Melnyk, 1972).

In our struggle for efficiency, however, we often sacrifice the convenience of the user for the convenience of the system. Users are forced to understand cryptic messages and respond with codes rather than with words. Somehow, we forget what it is like to be users the minute we become programmers, just as we forget what it is like to be pedestrians when we become drivers. Our programs then impart the character of a cold automaton rather than a human author.

The recommendations presented in the remainder of this paper are intended to provide preliminary guidelines for programmers who wish to rehumanize their programs but who have only a minimum computer interface (standard teletypewriter or small cathode-ray tube) and knowledge of a high level language such as BASIC. They are interpretations (both inductive and deductive) of related literature and should not be construed as recommendations specifically intended by the referenced authors in their original contexts.

A PRELIMINARY GUIDE TO STYLE IN INTERACTIVE PROGRAMMING

> As yet, no acknowledged sense of style has developed for CAI... In the meantime, however, some singularly unstylish CAI programs are being written. (Martin, 1973, p. 413)

(1) Maximize the amount of interaction in your programs.

Meridith (1973) suggests that "machine surrogate tutors" best impart information through continuous interaction. Foley (1973) points out that a program will have the best chance of understanding the user's desires if it can get the user to supply a "stream of input". Yntema (1969) has observed that as interactions become more "expensive" (both in the monetary sense and in the number allowed), computer users are far more anxious about making errors. Interaction can be maximized by keeping your messages short and requiring a user response after every few lines.

(2) Tie your programs in with other media.

Sometimes, short messages do not provide enough latitude to tell the user all that is necessary. But rather than print out several pages of text on the terminal, Heines (1975) suggests that a user's guide be written to accompany the program. The user's guide might include diagrams and photographs which do not lend themselves to computer display or just descriptions that are easier to read from a printed page than from the computer terminal.

(3) Use upper and lower case if available.

This recommendation has been made by Repko (1975), among others, and is consistent with her view that a computer system should "conform to the user's conception of the environment". As text is normally presented in upper and lower case, so should it be on the computer terminal (if physically possible). Upper case text has a cold, official feeling while lower case text is less forbidding.

(4) Display program output along the entire width of your screen or paper.

Gregory and Poulton (1970) found that poor readers had significantly poorer reading comprehension when text of seven words per line was right-justified as compared to their ability to comprehend the same material with uneven right-hand margins. (Good readers showed no significant difference in comprehension with the two methods of presentation.) This effect was nullified, however, when the text was lengthened to twelve words per line. The primary implication of this finding is that text of seven words per line or less should not be right-justified. The secondary implication is that text lines should be made as long as possible, at least up to twelve words per line. Thus the width of the output device (screen or paper) should limit the length of your message lines rather than the convenience of the program.

For example, consider the normal user of the PRINT statement to produce program output in BASIC. If the length of a statement line is limited to the width of the output medium, program output must be at least eight characters shorter than the width of the medium. This is because at least one space is needed for the line number, five for the PRINT command, and two for the opening and closing quotes. The problem can be easily solved by using a semicolon at the end of a PRINT statement and continuing the additional text with the next statement line.

(5) Keep format and style in mind.

McLaughlin (1966) compared the abilities of college undergraduates to locate information in well-produced and poorly produced (verbose) technical pamphlets. He found no significant difference in test performance when the two types of pamphlets were used by motivated students. Unmotivated students, however, showed significantly poorer performance when they used the poorly produced pamphlet as compared to their performance with the well-produced one. In both cases--motivated and unmotivated--students spontaneously stated that they would not have read the poorer version voluntarily. McLaughlin concludes:

> Objective measurement may show that the style of presentation of printed technical matter has little effect upon the efficiency with which information can be culled from it. Yet subjective preferences may be so strong as to make readers ignore material presented in a certain style. (Page 257)

(6) <u>Consider the experience of your target population</u>.

Mills (1967) and Nickerson (1969) have stated that the population of computer users is becoming increasingly heterogeneous. This means that more and more naive users are continually coming into contact with interactive computer programs. The style of these programs should therefore be friendly and conversational (Martin, 1973). Repko (1975) feels that programs written for naive users must not assume the programmer's knowledge of computer terms and operations. She acknowledges the difficulty of "putting yourself into your user's shoes" by describing the programmer as seeing the computer from the "inside" while the user sees it from the "outside".

Consider the act of entering data to a program in an interactive mode. In BASIC, the program statement used for this purpose is INPUT, which prints a question mark on the terminal and accepts data typed at the keyboard. Very often, therefore, one sees interactive programs which print out queries like this:

```
INPUT THE INTEREST RATE IN % PER YEAR
?
```

This query clearly reflects the programmer's view of the data entry procedure. The user must interpret the word "input" as "type". With little additional effort, the programmer can use the question mark as normal punctuation and relate more closely to the user's view of the data entry procedure:

```
WHAT IS THE INTEREST RATE IN % PER YEAR?
```

This is a simple question, and the fact that a user response is required is more obvious.

(7) Prompt the user as to the type of response required.

Even the most obvious query to an interactive programmer may not immediately indicate to a naive user the type of response to be made. When users are prompted, however, the doubt is quickly erased (Heines, 1974). For example,

> PLEASE TYPE "YES" OR "NO" IN RESPONSE TO THE FOLLOWING QUESTION AND THEN PRESS THE RETURN KEY:
>
> HAVE YOU EVER USED THIS TERMINAL BEFORE?

Once this type of instruction is given, shorter prompts usually suffice:

> WOULD YOU LIKE TO RUN THE PROGRAM AGAIN NOW ("YES" OR "NO")?

(8) Use menus to indicate the user's options.

Option menus have been used successfully with all classes of computer users (Foley, 1973; Martin, 1973). This technique allows the user to select an option from a given list quickly and efficiently because all available options are displayed and indication of the option desired by the user is extremely simple. Following is an example of a simple option menu which guides the student through an interactive environment (Heines, 1974):

> YOU ARE NOW REGISTERED FOR THIS TERMINAL SESSION AND MAY SELECT A PROGRAM OPTION FROM THE FOLLOWING LIST:
>
> (1) RUN A CHECK POINT PROGRAM
> (2) DISPLAY ALL THE DATA STORED ON YOUR WORK
> (3) DISPLAY ALL STORED DATA IN SUMMARY FORM
>
> OR...
>
> (4) END THIS TERMINAL SESSION
>
> WHICH OPTION WOULD YOU LIKE TO EXECUTE NOW (TYPE A NUMBER)?

(9) <u>Make error messages friendly and factual</u>.

Programmers very often overlook the importance of carefully constructed error messages. In the worse case, their error messages are flippant insults to the user. At best, error messages are often omitted, queries answered incorrectly are simply reprinted, and the user is surprised to see a question asked again that he or she has just answered. Consider the following interaction:

```
Computer: HAVE YOU EVER USED THIS TERMINAL BEFORE?
User    : YSE
Computer: HAVE YOU EVER USED THIS TERMINAL BEFORE?
```

The naive user will surely be confused, thinking that he or she has already responded "YES" to this query.

One solution is to tell the user that an incorrect response has been made. When this is done, however, Meredith (1973) suggests that messages using terms such as "not understood" or "rephrase" should be used rather than "that most irritating word in the programmer's lexicon: ILLEGAL!"

Foley (1973) stresses that "to the user, each error is a unique obstacle". The programmer must handle unanticipated responses "elegantly", he continues, encouraging the user to correct the error. He notes:

> A reference librarian is very unlikely to tell the user that she has no idea what he is talking about -- yet this is exactly what computer information systems regularly do. Thus they quickly gain a reputation for being frustrating.

The following example of error handling improves upon the interaction shown previously:

```
Computer: HAVE YOU EVER USED THIS TERMINAL BEFORE?
User    : YSE
Computer: I CAN ONLY RECOGNIZE THE RESPONSES "YES"
          OR "NO" TO THIS QUESTION.  PLEASE CHECK
          YOUR RESPONSE AND TRY AGAIN.

HAVE YOU EVER USED THIS TERMINAL BEFORE?
```

This type of explanatory error message can be sufficiently generalized to be programmed as a subroutine and called whenever a "yes" or "no" response is required.

(10) Do not eliminate message redundancy at the expense of message clarity.

Some readers may feel that the above message is far too wordy to be practical, especially if it is repeated each time this mistake is made. But the balance of message redundancy and clarity is often delicate: too much redundancy can bore an audience while too little can confuse them (Schramm, 1954). Nickerson (1969) admits that "all users tend to be impatient with redundant and non-informative messages", but further notes that:

> ...the extent to which any particular communication from the computer is redundant or non-informative depends upon the amount of experience that the user has had with the system.

For example, the message:

```
DA 90
```

may be sufficient for some users but non-informative for others. The message:

```
OUT OF DATA AT LINE 90
```

yields more information, but may be redundant for experienced users.

A solution suggested by Nickerson (1969) is to use shorter abbreviations and mnemonics, but to allow the user to view the longer, less cryptic message by entering, for example, "what" or "?". One should also consider the display rate of the user's terminal when planning error messages, as longer messages are tolerable when they are displayed quickly but intolerable if they are displayed slowly.

(11) Give the user as much feedback as possible.

Foley (1973) and Schramm (1954) have pointed out the importance of feedback for the successful operation of any communication system. Melnyk (1972) and Meredith (1973) have related the use of feedback to interactive computer programs in the form of error messages for incorrect input. But feedback can also keep the user informed of the state of the system. For example, naive users are often confused when the terminal pauses if input is not required, as may be the case while a tape or disk file is being processed. Confusion can be avoided by printing, for example:

YOUR SCORE IS NOW BEING RECORDED...

or, more simply:

ONE MOMENT, PLEASE...

(12) Use graphics wherever possible.

Graphics need not be limited to expensive, sophisticated systems. Even simple diagrams can be very helpful in trying to communicate ideas. While teletypewriters are extremely slow for displaying graphics, small cathode-ray tubes (CRT's) usually have special functions such as tab, backspace, and screen clear which can be used to speed up the rate at which graphics can be displayed. These features make it feasible to draw graphs and diagrams at a reasonable rate within the limitations of the terminal's character set.

(13) Write your programs so that they can be changed, improved, and enhanced.

As interactive programs are used, their strengths and weaknesses become apparent. If programmed in a haphazard manner, their weaknesses can be very difficult to correct, even if they involve only a small change in wording and structure. Repko (1975) suggests that programs be made flexible by functional division, isolating the input and output sections (see Figure 3). She comments:

> The mechanism of the program should never be an excuse for not allowing changes in the man-machine communication.

The simplest way to make programs adaptable is to document them extensively.

(14) Never give in to the machine.

Anyone who has asked a busy programmer for assistance on a programming problem has heard the reply, "It can't be done". Usually, this simply means that the problem appears to be non-trivial and the programmer does not wish to take the time to help you. Time and again, however, computer people have proven that there is some truth in the saying, "the impossible we do immediately; miracles take a little longer". No matter how impossible your idea might seem at

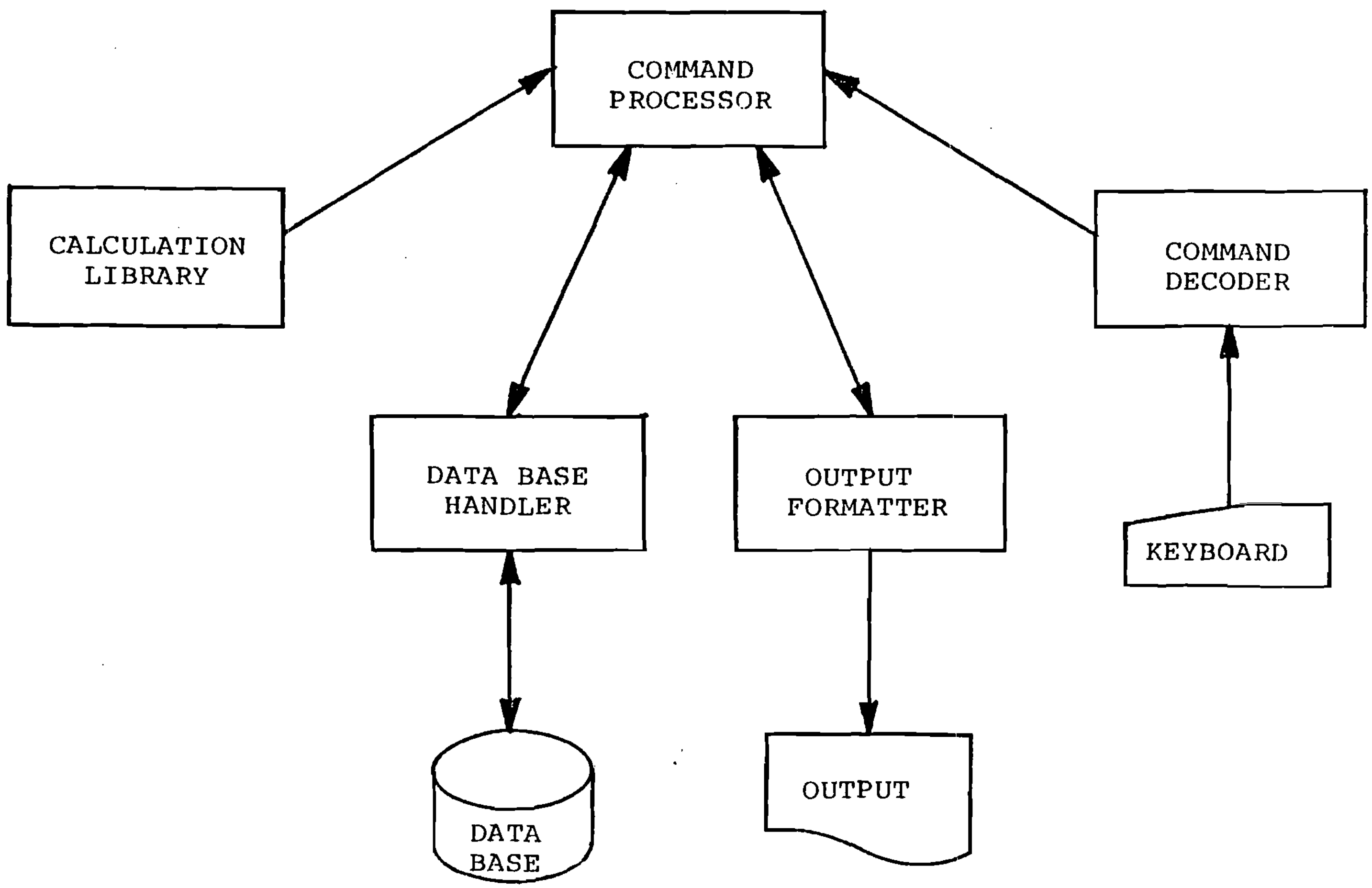


Figure 3

Functional Division of a Computer Program

[adapted from Repko (1975), page 7]

first, you can usually implement it in some form even if you have to compromise slightly. Careful scrutiny of your system will almost always reveal ways to get around limitations imposed by the hardware and software.

CONCLUSION

The quidelines presented in this paper are not novel. When viewed in retrospect, most of the recommendations appear to be the product of plain common sense. It is encouraging to note, however, that actual research does support these simple ideas. It is this author's hope that further research will expand this effort and that interactive programs written with a clear communicative style will make it easier for people from all backgrounds to use the computer effectively and enjoyably.

REFERENCES CITED

Chapinis, Alphonse. Prelude to 2001: explorations in human communication. American Psychologist 26(11):949-961, November 1971.

Foley, Joseph M. Communication aspects of information science. Theory Into Practice 12(3):167-172, June 1973.

Gregory, Margaret and E. C. Poulton. Even versus uneven right-hand margins and the rate of comprehension in reading. Ergonomics 13(4):427-434, July 1970.

Heines, Jesse M. An Interactive, Computer-Managed Model for the Evaluation of Audio-Tutorial Instruction. Unpublished Master's Thesis, University of Maine at Orono, May 1974.

__________. Everything you always wanted to know about computers ... you can teach yourself. Educational Technology 15(5):47-50, May 1975.

Martin, James. Design of Man-Computer Dialogues. Prentice-Hall, Inc., Englewood Cliffs, N.J. 1973, pp. 373-423.

McLaughlin, G. Harry. Comparing styles of presenting technical information. Ergonomics 9(3):257-259, May 1966.

Melnyk, Vera. Man-machine interface: frustration. Journal of the American Society for Information Science 23(6):392-401, November-December 1972.

Meredith, J. C. Machine as tutor. The Computer and Education. (The Educational Technology Review Series: Number Nine.) Educational Technology Publications, Englewood Cliffs, N.J. 1973, pp. 27-34.

Mills, R. G. Man-machine communication and problem solving. In C. A. Cuadra (ed.), Annual Review of Information Science and Technology, Volume 2. Interscience Publications, N.Y. 1967.

Nickerson, R. S. Man-computer interaction: a challenge for human factors research. Ergonomics 12(4):5Ø1-517, July 1969.

Repko, Marya. Man-machine communication. Decuscope 14(1):5-7, February 1975.

Schramm, Wilbur. How communication works. In Wilbur Schramm (ed.), The Process and Effects of Mass Communication. University of Illinois Press, Urbana, Illinois. 1954, pp. 3-1Ø.

Yntema, D. B. Engineering psychology in man-computer interaction. Paper presented at the Annual Convention of the American Psychological Association, San Francisco, California. 1969.