REPORT ON AN EXPERIMENT ON THE DESIGN OF INTERACTIVE COMMAND LANGUAGES

Henry Ledgard*

John Whiteside**

Andrew Singer***

William Seymour****

COINS Technical Report 79-21

Keywords: Human Engineering, Interactive Language, Command Languages, Psychology of Computer Use

CR Categories 3.36, 4.6

The work reported herein was sponsored by the National Science Foundation.

* Computer and Information Science Department, University of Massachusetts, Amherst, MA 01003

** Consultant in Psychology, 33 Guild Road, Worcester, MA 01602

*** E and L Instruments, Derby, CT 06798

**** Market Facts Inc., Los Angeles, CA 90024

TABLE OF CONTENTS

ABSTRACT

This report describes an experiment to test the hypothesis that natural language is a useful model for the human engineering of interactive command languages. The goal was to establish the assertion that a syntax employing familiar, descriptive, everyday words and well-formed English phrases contributes to a language that can be easily and effectively used. The experimental results strongly indicate that this assertion is valid.

1. General Philosophy

In the development of any technology, there is always a tendency to lose sight of the basic problems that stimulated its introduction. After the first flush of success, interest naturally tends to focus on the technology itself and the problems it presents. We have often forgotten that the original objective of computer technology was not just to develop more powerful systems but to increase the overall effectiveness of a human problem solver.

The work described in this paper has proceeded from the general assumption that experimental study of man-computer interaction can make an effective contribution to the design of interactive systems. Interacting with a computer is a complex and demanding intellectual activity. Some of the complexity is inherent in the nature of computing, but much results from the lack of design principles that are informed by knowledge of human capabilities, limitations, and preferences.

Importantly, any such principles, especially those that are not well accepted, should be supported by experimental evidence.

We consider here the design of interactive command languages, that is, any system where a user and a computer engage in a dialogue. Designing such a system involves many decisions about which commands to include, what sequences of operations to allow, and so on. Our concern is that there be deep rooted design principles motivated by concern for the convenience, efficiency, and comfort of the evenutal user.

2. Our Hypothesis

As one starting point for such research, we offer the speculation that natural language provides a suitable model for the design of interactive command languages. This hypothesis is not generally accepted, as evidenced by the number of interactive languages that frequently violate it.

The rationale for this hypothesis stems from a good deal of general empirical evidence, and has been explored by (Singer et al. 1978). A large number of psycho-linguistic experiments suggest that all aspects of human information processing are highly governed by natural language experience. Furthermore, of all languages one's natural language is certainly the most familiar of all. It makes sense to capitalize on this common human background.

One testable assertion that follows from this hypothesis that for native users: *an interactive system should be based on familiar descriptive, everyday words, and legitimate English phrases*. Our contention is that such systems will be easier to learn and use than a system that does not have these characteristics but that is similar in other respects.

Consider the following innocuous command

RS: /TOOTH/,/TRUTH/

taken from a typical interactive text editor. RS stands for "Replace String." The effect of the command is to replace the next occurrence of the string TOOTH to TRUTH.

As mentioned above, a large number of psycho-linguistic experiments seem to suggest that all aspects of human information

processing are highly trained by natural language experience. Drawing upon this experience, we believe that interactive languages should be closely modeled after natural language phrase structure to take advantage of such training. This is a format like

REPLACE "TOOTH" WITH "TRUTH"

or

CHANGE "TOOTH" TO "TRUTH"

(where the individual keywords may be abbreviated according to some convention) suggests our natural language experience.

To test this hypothesis, we chose a simple text editor as the object of our study. Editors are used by a cross section of users ranging from naive to professional. As generally experienced, more terminal time is spent editing than performing any other function. Our experiment involved a comparison of two text editors, identical semantically (in terms of editing power) but with differing syntax. One editor was a slightly modified version of the commercially available Control Data Corporation editor supplied with NOS. Its syntax does not resemble that of natural language. The second editor was a remodeled version of the NOS editor, with identical power but with its syntax altered so that its commands were all based on legitimate English phrases composed of common descriptive words.

The experiment was thus both a test of our hypothesis concerning natural language as an appropriate model of interactive languages, and also a demonstration of the effects that human engineering can have on commerically available software in terms of human efficiency, performance, and satisfaction.

3. The Experiment

3.1 Subjects

Twenty-four paid volunteers served as subjects. Equally represented by this group were three levels of familarity with interactive computing.

Group 1, termed "inexperienced users", consisted of eight individuals who claimed little or no experience with computer terminals (less than 10 hours of use).

Group 2, termed "familiar users", was composed of eight individuals who claimed between 11 and 100 hours of experience using a computer terminal.

Group 3, termed "experienced users", consisted of eight subjects who claimed over 100 hours of terminal use.

As a requirement for participation, all subjects had to be able to type a presentable college term paper.

3.2 The Editors

Two text editors were used, termed "the notational editor" and "the English editor." The notational editor was modeled after the Control Data Corporation NOS Version I Text Editor. The available commands in the notational editor formed a subset of those found in the NOS editor. The English editor contained the same logical requests but with a syntax dictated by the requirement that all commands be based on legitimate English

phrases formed of familiar descriptive words. Table 1 summarizes the syntax rules for the so-called "notational" editor. Table 2 summarizes the revised syntax for the so-called "English" editor.

Subjects were given Tables 1 and 2, as well as detailed manuals describing each editor. The subjects saw the editors labeled Editor X and Editor Y, not by the names used in this report.

3.3 Procedure

The subjects were informed about the general nature of the experiment, that it would involve text editing and then actually performing some editing tasks. They were informed of their rights as human subjects in research and asked to sign a statement of informed consent. Once a determination had been made concerning the number of hours of terminal experience each subject had, he or she was assigned to either the inexperienced, familiar, or experienced group.

Each subject was then given one of the manuals and a table of sample editing commands. Subjects were free to study these materials for as long as they needed and the experimenter answered any questions. The experimenter then demonstrated the use of the terminal and one editing command. At this point the subject was encouraged to practice using the editor until he or she felt comfortable with it. Practice editing was done using a short listing of world gold prices as text.

TABLE 1 Summary of Notational Editor

Example Commands	Function
FIND	Current line moves ahead 1 line.
FIND;5	Current line moves ahead 5 lines.
FIND;*	Current line moves ahead to last line in text.
FIND:/TOOTH/	Current lines moves ahead to last line in text containing "TOOTH."
FIND;-1	Current line moves back 1 line.
FIND;-3	Current line moves back 3 lines.
FIND;-*	Current line moves back to first line in text.
FIND:/TOOTH/;-1	Current line moves back to nearest line containing "TOOTH."
LIST	Current line is displayed.
LIST;10	Displays 10 lines starting with current line. New current line becomes last line displayed.
LIST;*	Displays all lines from the current line to the last line of text. Current line becomes last line displayed.
LIST:/KO/;*	Displays all lines with "KO" that are at or past current line. Current line becomes last line displayed.
DELETE	Erases the current line, next line becomes current line.
DELETE;7	Erases 7 lines starting with current line. Next line becomes current line
EXTRACT	Puts current line in temporary buffer. Does not change current line.
EXTRACT;8	Puts 8 lines starting with current line in temporary buffer. Current line becomes last line in buffer.
ADD	Computer types two plus signs. Anything typed after is inserted into text after current line. To stop inserting type dual character "Ctrl-T". Current line is then last line you typed.
ADD $	In response to ADD, computer displays two plus signs. Typing "$" inserts contents of buffer into text just after current line. New current line becomes last line from buffer.
CHANGE	Computer erases current line, then types two plus signs. Anything now typed is inserted in place of erased line. Type "Ctrl-T" to stop inserting. Current line is then last line you typed.
CHANGE $	In response to CHANGE, computer displays two plus signs. Typing the "$" character erases the current line and replaces it with the buffer. Current line becomes last line from buffer.
RS:/KO/,/OK/	Searches current and all subsequent lines for first "KO" and changes it to "OK". Current line becomes line that was changed.
RS:/KO/,/OK/;*	Changes every instance of "KO" to "OK" in current and all subsequent lines. Current line becomes last line to be changed.

IMPORTANT: All commands can be abbreviated to the first letter or each word. It is not necessary to insert spaces between the letters of abbreviated commands. Thus

LIST:/TOOTH/;*

can be written

L:/TOOTH/;*

or as

L : /TOOTH/ ; *

TABLE 2 Summary of English Editor

EXAMPLE COMMANDS	FUNCTION
FORWARD	Current line moves ahead 1 line.
FORWARD 5 LINES	Current line moves ahead 5 lines.
FORWARD ALL LINES	Current line moves ahead to last line in text.
FORWARD TO "TOOTH"	Current line moves ahead to next line containing "TOOTH."
BACKWARD	Current line moves back 1 line.
BACKWARD 5 LINES	Current line moves back 5 lines.
BACKWARD ALL LINES	Current line moves back to first line in text.
BACKWARD TO "TOOTH"	Current line moves back to nearest line containing "TOOTH."
LIST	Current line is displayed.
LIST 10 LINES	Displays 10 lines starting with current line. New current line becomes last line displayed.
LIST ALL LINES	Displays all lines from the current line to the last line of text. Current line becomes last line displayed.
LIST ALL LINES WITH "KO"	Displays all lines with "KO" that are at or past current line. Current line becomes last line displayed.
DELETE	Erases the current line, next line becomes current line.
DELETE 7 LINES	Erases 7 lines starting with current line. Next line becomes current line.
HOLD	Puts current line in temporary holder. Does not change current line.
HOLD 8 LINES	Puts 8 lines starting with current line in temporary holder. Current line becomes last line in holder.
INSERT TEXT AFTER	Computer types two question marks. Anything typed after is inserted into text after current line. To stop inserting type dual character "Ctrl-T". Current line is then last line you typed.
INSERT TEXT OVER	Computer erases current line, then types two question marks. Anything now typed is inserted in place of erased line. Type "Ctrl-T" to stop inserting. Current line is then last line you typed.
INSERT HOLDER AFTER	Inserts contents of temporary holder into text just after current line. Current line becomes last line from holder.
INSERT HOLDER OVER	Erases current line and replaces it with contents of holder. Current line becomes last line from holder.
CHANGE "KO" TO "OK"	Searches current and all subsequent lines for first "KO" and changes it to "OK". Current line becomes line that was changed.
CHANGE ALL "KO" TO "OK"	Changes every instance of "KO" to "OK" in current and all subsequent lines. Current line becomes last line to be changed.

IMPORTANT: All commands can be abbreviated to the first letter of each word. It is not necessary to insert spaces between the letters forming abbreviated commands. Thus

LIST ALL LINES WITH "TOOTH"

can be written

L A L W "TOOTH"

or as

LALW"TOOTH"

Following the practice, each subject was given a written copy of some text material adapted from Moody's Industrial Manual. Superimposed on this written copy were a set of proofreader-like marks indicating changes in the text that the subject was to make.

The subject was told that he or she had 20 minutes to make changes and would be scored according to how quickly and accurately the changes were made. The experimenter then allowed the subject to start the task when ready.

Following the completion of 20 minutes of editing, each subject was given the materials for the editor that had not been used previously. Again the subject was allowed to study the materials, ask questions, and practice the commands on the practice text. Then a second text adapted from Moody's Manual was provided, similar in all respects to the first except for the specific content and required changes. Again each subject was given 20 minutes to perform the editing task.

3.4 Testing for Bias

Before the experimental session, each subject was administered the preference questionnaire shown in Table 3.

This questionnaire gives three pairs of commands. One member of each pair was modeled after the notational editor and one after the English editor. Subjects were asked to give their preference for one or the other form of commands. A

TABLE 3 Preference Questionnaire

Below are three pairs of editor commands. Each pair accomplishes the same function even though members of a pair look different. In each case, please indicate which pair member you would prefer to use and how strongly you feel about it using the rating scale beside the commands. Circle the number that best expresses your feelings.

1. This pair of commands moves a line pointer back one line.

   Command A: BACKWARD

   Command B: FIND;-1

   1. Strongly prefer Command A
   2. Mildly prefer Command A
   3. No preference
   4. Mildly prefer Command B
   5. Strongly prefer Command B

2. This pair of commands changes instances of the word "TOOTH" to "TRUTH" within a line.

   Command A: CHANGE "TOOTH" TO "TRUTH"

   Command B: RS: | TOOTH | , | TRUTH |

   1. Strongly prefer Command A
   2. Mildly prefer Command A
   3. No preference
   4. Mildly prefer Command B
   5. Strongly prefer Command B

3. This pair of commands moves a line pointer back to the first line containing the word "TOOTH."

   Command A: FIND: | TOOTH | ;-1

   Command B: BACKWARD TO "TOOTH"

   1. Strongly prefer Command A
   2. Mildly prefer Command A
   3. No preference
   4. Mildly prefer Command B
   5. Strongly prefer Command B

rating of "1" indicated a strong preference for the English-language command, "3" indicated indifference, and "5" showed a strong preference for the corresponding notational command.

After the experimental session, the subjects were asked to state their preference for the notational versus the English editor.

4. Quantitative Analysis

The dependent measures taken in the experiment were:

1. A calculation of editing efficiency
2. The percentage of erroneous commands
3. The percentage of the editing task completed.

The independent variables were:

1. Type of editor: two levels
2. Amount of terminal experience: three levels.

The measure of editing efficiency was calculated as follows. Let

POS be the number of commands that resulted in an improvement of the text
NEG be the number of commands that resulted in a degradation of the text
NUM_CMDS be the total number of commands issued

EDITING_EFFICIENCY = (POS - NEG)/NUM_CMDS

The percentage of erroneous commands was calculated as follows. Let

SYN be the number of commands that were syntactically ill-formed
SEM be the number of commands that were semantically meaningless
NUM_CMDS be the total number of commands issued

ERROR_RATE = (SYN + SEM)/NUM_CMDS

The percentage of the editing task completed was calculated as follows. Let

COR be the number of indicated corrections made to the text
ERR be the number of erroneous changes made to the the text
TOT_COR be the total number of indicated corrections requested

COMPLETION_RATE = (COR - ERR)/TOT_COR

All measures were calculated by hand.

5. Results

Table 4 summarizes the results of the experiment.

Percentage of task completed. Overall the subjects were only able to complete 48% of the editing task using the notational editor as opposed to 63% using the English editor. This difference is statistically significant at better than the .001 level, which means there is less than one chance in a thousand that this difference is due to randomized effects. Furthermore, the English editor outperformed the notational editor regardless of the level of experience of the users.

The interaction of task order and editor type was statistically significant. This means that, above and beyond the overall difference between editor types, subjects tended to do better on whichever editor they used second.

Experience affected the amount of editing work done; the inexperienced subjects as a group were only able to complete 35% of the task, whereas the experienced users finished 79%. This effect of experience was also statistically significant at better than the .001 level.

TABLE 4 Summary of Performance Results

(a) Percentage of Editing Task Completed

	English Editor	Notational Editor	Average Across Editors
Inexperienced Users	42%	28%	35%
Familiar Users	63%	43%	53%
Experienced Users	84%	74%	79%
Average Across Users	63%	48%	

(b) Percentage of Erroneous Commands

	English Editor	Notational Editor	Average Across Editors
Inexperienced Users	11%	19%	15%
Familiar Users	6.4%	18%	12%
Experienced Users	5.6%	9.9%	7.8%
Average Across Users	7.8%	16%	

(c) Editing Efficiency

	English Editor	Notational Editor	Average Across Editors
Inexperienced Users	43%	31%	37%
Familiar Users	53%	36%	44%
Experienced Users	58%	53%	55%
Average Across Users	51%	40%	

Erroneous commands. Disregarding all other factors, the error rate for the English editor was 7.8%, whereas the rate for the notational editor was twice this, at 16%. The analysis of variance performed on the error data shows that the chance that the two editors do not, in fact, differ with respect to error rates is less than .001. Table 4 suggests that fewer errors were made by more experienced users, but the effect of experience was not significant at the .05 level.

As with the completion data, the absence of a significant interaction between level of experience and editor type means that more errors were committed using the notational editor regardless of the level of experience.

Editing efficiency. Again the two editors were significantly different from one another, with the English language editor being used at 51% efficiency as opposed to 40% efficiency for the notational editor. On average, 1.96 commands were required to produce a single editing change using the English editor, whereas 2.51 commands were required using the notational editor.

The more experienced users were able to make more efficient use of both editors, as shown by the significant effect of experience.

Preference. A measure of which command set the users preferred was taken both before and after using the editors. In the case of the familiar and experienced users, both before and after use; in the case of the inexperienced users, only after use. Table 5 shows the results. A score of 1 indicated strong preference for the English commands, 3 no preference, and 5 a strong preference for the notational commands.

All groups clearly preferred the English language editor after exposure to both, but the experienced users showed less of a preference than the other two groups. Prior to using the editors. However, the familiar group had a slight preference for the notational command set; the experience of using both seems to have caused considerable change of attitude. The experienced group had a slight preference for the English commands prior to editing and changed their preference slightly more in favor of the English commands after completing the editing tasks.

TABLE 5 Results of Preference Questionnaire

	Before Editing	After Editing
Inexperienced Users	--	1.25
Familiar Users	3.25	1.25
Experienced Users	2.75	2.13

Key:

1. Strongly prefer English editor
2. Mildly prefer English editor
3. No preference
4. Mildly prefer notational editor
5. Strongly prefer notational editor

6. Implications

The results demonstrate that redesigning the surface syntax of a commercial editor so that the commands more closely resemble English phrases resulted in far better performance. On all measures, performance using the English editor was superior to performance using the notational editor. This was true regardless of the experience level of the users.

Besides being statistically significant, the results are striking in absolute terms. Across all users, the completion rate using the English editor versus the notational editor was at a ratio of 63 versus 48 percent, and the editing efficiency at 51 versus 40%. Furthermore, nearly twice as many errors were made with the notational editor. These differences were even sharper among inexperienced and familiar users.

The editors were identical in editing power (semantics) and differed only in the appearance of the actual commands (syntax). In a sense, the English editor is only a variation of the notational; both are basically the same editor. Yet the performance differences were striking. It appears that the surface syntax of a language is quite important from a human engineering point of view.

In the course of running the experiment, the experimenter (Whiteside) was struck by the observation that *the users made no distinction between syntax and semantics*. They simply

could not conceive of editing power or function as something different from the appearance of the actual commands. To them, the actual commands embodied the editor to such an extent that many were surprised when told after the experiment that the two editors were functionally identical. This suggests that language designers must be as much concerned with surface syntax as with functional features if they hope to design a product to optimize performance.

As mentioned earlier, we were concerned that, since each user was to use both editors, the user bias might influence the results. This was the reason for the preference questionnaire. As a whole, the familiar and experienced users showed no bias toward one or the other set of commands prior to using the editors. At the conclusion of the experiment, 22 of the 24 users preferred the English editor. The two subjects who did not were both experienced users. In any event, pre-experimental bias is an unlikely explanation of the results.

The users had unlimited time to practice with the editors. This too might produce a bias for one editor over the other. However, the average number of commands issued during practice with the English editor was 14.4 as opposed to 14.7 commands with the notational editor. The difference is not statistically significant and seems so small that different amounts of practice can be ruled out as an explanation of the results.

It is generally accepted that training is more effective with human assistance. As a result, the experimenter was present to answer questions during learning and practice with the editors. We were concerned here too that the experimenter might introduce a bias. To check this, audio tapes were made of the sessions with two subjects; the experimenter was not aware of the actual purpose of the tapes. Here again, we could not detect any bias.

A logical extension of our study would be the examination of asymptotic performance: Would the performance differences remain if users continued to use the editors for a long time? Clearly performance on both editors would improve, and the performance differences would probably decrease. But this would not alter the implications of our result for interactive software design.

First, humans can clearly adapt themselves to difficult systems including ones that have been poorly human engineered. The question is, at what cost? With years of practice, people can learn to transmit Morse code at a rate comparable to average two-finger-typing speed, but no one argues that Morse code senders should replace typewriter keyboards. Why burden users with systems that are difficult to learn when better ones are available?

Second, in our experiment there were performance differences between the two editors even for users with a large amount of interactive computing and editing experience. All of these experienced users were well versed with two computer

text editors, and most knew at least three. Yet when faced with new editors, they exhibited the same performance differences as individuals who were just learning to use a computer terminal. We feel that the experienced user when faced with a new system or with a system that he or she uses only intermittently is in much the same position as the novice user, and so needs good human engineering just as much.

To test these ideas even further, we performed a small post-experiment study. We recurited four users who had at least 50 hours experience with the notational NOS editor, but no experience with the English editor. With these subjects no significant performance differences were found, even though the English editor alone was new to these users. We take this small pilot result as again strong confirmation of our hypothesis.

In interactive software design too little attention is paid to the need, limitations, and capabilities of the end user. We have shown that a relatively simple change in the syntax of a computer text editor could result in a much improved product. We believe that our specific result deserves to have impact on the design of future interactive software systems.

Acknowledgment

Michael Marcotty provided thoughtful advice throughout the design of this experiment. We are also grateful Rich Shire, Daryl Winters, Lance Miller, John Gannon, Chuck Clifford, Dan Anderson, Ram Dahiya, and Conrad Wogrin, who assisted us in various ways.

APPENDIX 1

More Detailed Statistical Results

The following table gives a detailed summary of the statistical results. It uses the following conventions.

df This is the number of degrees of freedom.

SS This is sum of squares giving the square deviation attributed to source of variance.

MS This is mean square deviation.

F This is the F ratio, the result of a statistical test to determine significance.

** Indicates that a confidence level better than .01 was obtained.

*** Indicates that a conficence level better than .001 was obtained.

(a) *Percentage of Editing Task Completed*

Source of Variance	df	SS	MS	F
Total	47	3.041		
Between Subjects	23	2.514		
Task Order	1	.020	.020	<1
Experience Level	2	1.556	.778	14.962***
Order by Experience Interaction	2	.008	.004	<1
Error Between Subjects	18	.930	.052	
Within Subjects	24	.527		
Editor Type	1	.266	.266	30.96***
Editor by Order Interaction	1	.082	.082	9.53**
Editor by Experience Interaction	2	.023	.012	1.33
Editor by Experience By Order Interaction	2	.001	.0005	<1
Error within Subjects	18	.154	.009	

(b) Percentage of Erroneous Commands

Source of Variance	df	SS	MS	F
Total	47	.4198		
Between Subjects	23	.2292		
Task Order	1	.0055	.0055	<1
Experience Level	2	.0452	.0226	2.3399
Order by Experience Interaction	2	.0033	.0017	
Error Between Subjects	18	.1752	.0097	
Within Subjects	24	.1906		
Editor Type	1	.0739	.0739	13.82**
Editor by Order Interaction	1	.0009	.0009	<1
Editor by Experience Interaction	2	.0100	.0050	<1
Editor by Experience By Order Interaction	2	.0096	.0048	<1
Error Within Subjects	18	.0962	.0053	

(c) Editing Efficiency

Source of Variance	df	SS	MS	F
Total	47	1.083		
Between Subjects	23	.680		
Task Order	1	.040	.040	1.90
Experience Level	2	.266	.133	6.49**
Order by Experience Interaction	2	.005	.0025	<1
Error Between Subjects	18	.369	.021	
Within Subjects	24	.403		
Editor Type	1	.153	.153	14.27**
Editor by Order Interaction	1	.010	.010	<1
Editor by Experience Interaction	2	.030	.015	1.364
Editor by Experience by Order Interaction	2	.017	.009	<1
Error Within Subjects	18	.193	.011	

** $p < .01$

*** $p < .001$

APPENDIX 2

Informed Consent Form

You are asked to participate in an experiment concerned with comparing two similar computer text editors. You will be asked to familiarize yourself with the operation of a computer terminal (if you are not familiar already) and then to study a manual describing one of the text editors. You will then be asked to perform a short editing task for practice. Once you have practiced you will be asked to do a longer editing task during which your performance will be measured. Finally you will be asked to study, practice, and perform another editing task with the second of the two editors.

The experimenter will assist you and answer any questions you have. You are completely free to stop participating in the experiment at any time. Any data concerning your performance will be held in strictest confidence.

The total time involved should be less than two hours. At the end of the experimental session you will be paid $10.00 for your participation.

If you are willing to participate, please sign the following statement:

"I have read the above description of experimental procedure and of my rights as a subject and I have agreed to participate."

Signed:

Date:

APPENDIX 3

General Instructions

A computer text editor is a software system designed to allow you to write and change textual material. It is the electronic equivalent of pencil, paper, sissors, and scotch tape. Using an editor allows you to give commands that create, change, erase, and locate portions of text. For example, suppose the computer has stored the following line of text:

YOU MUST TELL THE TOOTH

You want to change "TOOTH to "TRUTH" and so make a sensible statement. Using Editor X which you will be studying shortly you would type:

CHANGE "TOOTH" TO "TRUTH"

and the computer would automatically change the line to read:

YOU MUST TELL THE TRUTH

If you were using Editor Y the same result would be achieved by typing:

RS:/TOOTH/,/TRUTH/

As you will see shortly, the editors you will be using each consists of a set of commands similar to those shown above. Each command causes the editor to perform a specific task. In this experiment you will be given a copy of a section of text that is stored in the computer. The text contains a number of errors that you are to correct using the editing commands.

Now look at the summary of commands for the editor you will be using first. The experimenter will demonstrate the use of several of the commands and then give you time to study and practice until you feel comfortable. Try to practice each command at least once. Notice that you have both a brief summary of commands and also a more detailed manual. The brief summary should be consulted first and the more detailed manual can be looked at to answer any questions.

APPENDIX 4

Manual for the Notational Editor

The Current Line

At all times, one line of the text in the computer's memory is considered the current line. All commands use this line as a reference. For example, if you enter:

```
FIND;5
```

the computer understands this to mean "select a new current line 5 lines ahead of the present current line." Or if you say:

```
LIST
```

this is understood to mean "display the current line".

The current line is always the last line that the computer has printed. All commands except "LIST" will cause the computer to print the message:

```
THE CURRENT LINE IS
```

followed by the current line. In the case of the "LIST" command no message is printed but the new current line is nonetheless the last line printed.

The current line may be changed by the use of the commands:

1) FIND or FIND;-1

The above commands move the current line ahead or back one line.

To move forward or back any number of lines use:

2) FIND;n or FIND;-n

Where "n" is a number such as 3 or 9.

To move the current line ahead to the very last line of text or back to the very first line of text use:

3) FIND;* or FIND;-*

It is possible to move forward or back to a line identified by textual material that it contains. In response to the commands:

4) FIND:/text/ or FIND:/text/;-1

the computer will move ahead or back to the nearest line that contains within it the character string "text". This line will be displayed and will become the new current line.

In all cases the computer will type the new current line in response to these commands.

Displaying Lines

Often you will want the computer to display lines of text for you in order to refresh your memory and to keep you aware of changes you have made. There are four variations of the basic command to accomplish this:

To cause the current line to be displayed enter the command:

1) LIST

Note that this does not change the current line.

To cause any number of lines to be displayed use the command:

2) LIST;n

where "n" is any number such as 1,2, or 5. This will cause n lines to be displayed starting with the current line. Note that following completion of this command the current line will have changed and will now be the last line printed.

To display all lines from the current line to the last line in the text use the command:

3) LIST;*

It is also possible to cause the computer to display lines that contain a certain string of characters. In response to the command:

4) LIST:/text/;*

the computer will search for and display any lines that contain the character string "text". A character string is any combination of characters (keystrokes) including spaces. At the completion of this command the current line is the last line that the computer has typed.

Often it will be necessary to use combinations of commands to achieve a desired result. For example, suppose you want to display the entire text but the current line is not the first line of text. In this case the following sequence of commands is required:

```
FIND;-*
LIST;*
```

Deleting Lines

The deletion command will change the text as stored in the computer's memory. A single line or any number of lines may be deleted using a single command.

To delete the current line enter:

1) DELETE

To delete n lines starting with the current line enter:

2) DELETE;n

When the command has been executed the new current line will be the next line after the last line that was deleted.

Adding Lines

It is possible to insert new material into the text. To do this enter:

```
ADD
```

In response to this the computer will display two plus signs: "++", to indicate that it is ready to accept the new material. At this point type the material to be inserted. Any number of lines can be inserted using this command, in fact, any character typed will be interpreted as new text with the exception of the combination character "Ctrl-T". This special character is used to indicate that all the material to be inserted has been entered. Following the "Ctrl-T" character, the computer is again ready to accept new commands. "Ctrl-T" is formed by simultaneously pressing the key

labeled "Ctrl" and the "T" key, followed by pressing the "return" key.

The new material is inserted after the current line. Once the insertion is complete, the new current line is the last new line entered.

Changing Lines

The command:

CHANGE

operates identically to the "ADD" command except that the new material replaces the current line rather than appearing after it.

Moving Lines

It is possible to lift an entire section of text out of one location and to transfer it to another part of the text - much like snipping part of a page out with scissors and pasting it onto another page.

To specify the portion of text to be moved enter:

1) EXTRACT

or

2) EXTRACT;n

The EXTRACT command causes 1 or n lines (respectively) starting with the current line to be placed in a temporary buffer. Note that this command alone does not change the text in any way, nor does it *change* the current line.

Assuming that the buffer has been filled by an "EXTRACT" command, it can be inserted into another portion of the text. To do this requires the ADD and CHANGE commands discussed above and the symbol "$" which stands for the buffer.

Recall that typing ADD or CHANGE causes the computer to display two plus signs indicating that it is ready to accept text. Typing:

$

in response to the plus signs causes the buffer to be inserted either after or in place of the current line. Once the computer has responded to the "$" command, the current line is the last line of the buffer and the computer is ready to accept new commands.

To accomplish transfering a portion of text using these commands several steps are necessary. As an example, suppose the current line is to be moved so that it will come after the next line. The appropriate commands would be:

EXTRACT	(this fills the buffer with the current line)
DELETE	(the line to be moved is still in the text so it must be removed; the next line is now the current line)
ADD	(computer responds with "++")
$	(this inserts the line to be moved after the current line)

Changing text within a line

All the commands discussed so far operate on entire lines. It is also possible to alter only portions of a line or lines.

The command:

1) RS:/old-text/,/new-text/

will search for the first occurence of the character string "old-text" within any line starting with the current line. It will then delete this character string and replace it with whatever has been typed as "new-text".

The command:

2) RS:/old-text/,/new-text/;*

works in the same way except that every instance of "old-text" starting with the current line is altered to "new-text".

At the completion of the CHANGE command, the new current line will be printed; it will be the last line that the computer has changed.

IMPORTANT

All of the commands above can be abbreviated to contain only the first letters of words. In addition, the presence or absence of spaces between the letters is not important. Thus:

```
LIST:/text/;*
```

can be written as:

```
L:/text/;*
```

or even as:

```
L : / text / ; *
```

APPENDIX 5

Manual for the English Editor

The Current Line

At all times, one line of the text in the computer's memory is considered the current line. All commands use this line as a reference. For example, if you enter:

FORWARD 5 LINES

the computer understands this to mean "select a new current line 5 lines ahead of the present current line." Or if you say:

LIST

this is understood to mean "display the current line".

The current line is always the last line that the computer has printed. All commands except "LIST" will cause the computer to print the message:

THE CURRENT LINE IS

followed by the current line. In the case of the "LIST" command no message is printed but the new current line is nonetheless the last line printed.

The current line may be changed by the use of the commands:

1) FORWARD or BACKWARD

The above commands move the current line ahead or back one line.

To move forward or back any number of lines use:

2) FORWARD n LINES or BACKWARD n LINES

Where "n" is a number such as 3 or 9.

To move the current line ahead to the very last line of text or back to the very first line of text use:

3) FORWARD ALL LINES or BACKWARD ALL LINES

It is possible to move forward or back to a line identified by textual material that it contains. In response to the commands:

4) FORWARD TO "text" or BACKWARD TO "text"

the computer will move ahead or back to the nearest line that contains within it the character string "text". This line will be displayed and will become the new current line.

In all cases the computer will type the new current line in response to these commands.

Displaying Lines

Often you will want the computer to display lines of text for you in order to refresh your memory and to keep you aware of changes you have made. There are four variations of the basic command to accomplish this:

To cause the current line to be displayed enter the command:

1) LIST

Note that this does not change the current line.

To cause any number of lines to be displayed use the command:

2) LIST n LINES

where "n" is any number such as 1,2, or 5. This will cause n lines to be displayed starting with the current line. Note that following completion of this command the current line will have changed and will now be the last line printed.

To display all lines from the current line to the last line in the text use the command:

3) LIST ALL LINES

It is also possible to cause the computer to display lines that contain a certain string of characters. In response to the command:

4) LIST ALL LINES WITH "text"

the computer will search for and display any lines that contain the character string "text". A character string is any combination of characters (keystrokes) including spaces. At the completion of this command the current line is the last line that the computer has typed.

Often it will be necessary to use combinations of commands to achieve a desired result. For example, suppose you want to display the entire text but the current line is not the first line of text. In this case the following sequence of commands is required:

BACKWARD ALL LINES
LIST ALL LINES

Deleting Lines

The deletion command will change the text as stored in the computer's memory. A single line or any number of lines may be deleted using a single command.

To delete the current line enter:

1) DELETE

To delete n lines starting with the current line enter:

2) DELETE n LINES

When the command has been executed the new current line will be the next line after the last line that was deleted.

Adding Lines

It is possible to insert new material into the text. To do this enter:

INSERT TEXT AFTER

In response to this the computer will display two question marks: "??", to indicate that it is ready to accept the new material. At this point type the material to be inserted. Any number of lines can be inserted using this command, in fact, any character typed will be interpreted as new text with the exception of the combination character "Ctrl-T". This special character is used to indicate that all the material to be inserted has been entered. Following the "Ctrl-T" character, the computer is again ready to accept new commands. "Ctrl-T" is formed by simultaneously pressing the key

labeled "Ctrl" and the "T" key, followed by pressing the "return" key.

The new material is inserted after the current line. Once the insertion is complete, the new current line is the last new line entered.

Changing Lines

The command:

INSERT TEXT OVER

operates identically to the "INSERT TEXT AFTER" command except that the new material replaces the current line rather than appearing after it.

Moving Lines

It is possible to lift an entire section of text out of one location and to transfer it to another part of the text - much like snipping part of a page out with scissors and pasting it onto another page.

To specify the portion of text to be moved enter:

1) HOLD

or

2) HOLD n LINES

The HOLD commands cause 1 or n lines (respectively) starting with the current line to be placed in a temporary holder. Note that this command alone does not change the text in any way, nor does it change the current line.

Assuming that the holder has been filled by a "HOLD" command, the commands:

1) INSERT HOLDER AFTER

and

2) INSERT HOLDER OVER

will insert the contents of the holder either after or in place of the then current line.

To accomplish transfering a portion of text using these commands several steps are necessary. As an example, suppose the current line is to be moved so that it will come after the next line. The appropriate commands would be:

```
HOLD                  (this fills the holder with the
                       current line)
DELETE                (the line to be moved is still in
                       the text so it must be removed;
                       the next line is now the current
                       line)
INSERT HOLDER AFTER   (this inserts the line to
                       be moved after the current line)
```

Changing text within a line

All the commands discussed so far operate on entire *lines*.

It is also possible to alter only portions of a line or lines.

The command:

1) CHANGE "old-text" TO "new-text"

will search for the first occurence of the character string "old-text" within any line starting with the current line. It will then delete this character string and replace it with whatever has been typed as "new-text".

The command:

2) CHANGE ALL "old-text" TO "new-text"

works in the same way except that every instance of "old-text" starting with the current line is altered to "new-text".

At the completion of the CHANGE command, the new current line will be printed; it will be the last line that the computer has changed.

IMPORTANT

All of the commands above can be abbreviated to contain only the first letters of words. In addition, the presence or absence of spaces between the letters is not important. Thus:

LIST ALL LINES WITH "text"

can be written as:

L A L W "text"

or even as:

LALW"text"

APPENDIX 6

Sample Editing Task

(TAKEN FROM REPORST FILED WITH THE SECURITY AND EXCHANGE COMMISSION)
COMPARATIVE CONSOLIDATED INCOME ACCOUNT, YEARS ENDING DEC. 30:
(IN ~~HUNDREDS~~ THOUSANDS OF DOLLARS)

	'787	'76	'75
SALES	$2,325,780	$2,932,532	$2,458,050
COST OF SALES	2,870,878	2,574,670	2,287,800
SELLING & ADMIN. EXP	228,882	203,220	274,700
INTERETS INCOME	56,927	22,782	28,462
OTHER INCOME, NET	2,603	28,499	7,308
INTEREST EXPENSE	38,230	32,950	22,250
INCOME BEF. TAXES	296,420	252,080	99,070
INCOME TAXES	87,077	56,476	30,205
EXTR. ITEMS NET OF TAXES		-3,500	
DEFFERED TAXES	5,756	22,442	22,033
INVEST. TAX CREDIT	22,023	24,268	7,058
INCOME BEF. EXTR. ITEMS	225,620	97,330	36,890
LOSS FROM OPER			
RETAINED EARNINGS	487,328	409,206	363,327
COMM. DIVIDENDS	32,740	22,450	28,048
PREF. DIVIDENDS	256	256	256
OTHER		98	
NET INCOME	225,620	200,830	63,890
SUPPLIMENTARY DATA			
RETAINED EARNINGS	570,942	487,328	409,296
REPAIRS	222,072	99,672	72,275
DEPRECIATION	84,270	72,420	56,220
TAXES	32,446	25,707	32,935
RENTS		25,475	25,490

APPENDIX 7

Diary of Bill Seymour Continued

(Entries by Henry Ledgard)

December 1978

I have been away for some time working on the Honeywell design team on the DOD Common Language effort. I asked Daryl Winters to pretest a few subjects using Bill Seymour's work. Some subjects never finished the editing tasks, others showed no significant overall performance differences. The detailed listings show some sign of hope, but clearly, something is definitely wrong.

July 1979

Andrew Singer and I discuss what to do. We need help. He mentions meeting John Whiteside, a Ph.D. in Psychology, who has taken an interest in this area. Hope reappears, Whiteside is available.

(Entries by John Whiteside)

July 17, 1979

I am John A. Whiteside, a Master's student in computer science and an experimental psychology Ph.D. with 12 years' research experience. I met today with Henry Ledgard about completing this experiment, evidently started in June, 1977.

The goal agreed upon was the redesign, completion, and publication of an experiment based upon that described in the earlier entries of this diary. The experiment is to be designed so as to afford the greatest possible chance of finding significant differences in ease of use as a function of editor grammar. At the same time, the highest standards of methodological excellence and freedom from experimental bias are to be observed.

Although not indicated in the diary, evidently, six pilot subjects were tested. It appears that there was little evidence for differences between the grammars on the basis of overall time to task completion. Henry felt that an effect was present but was being masked by some defect in the experimental design, possibly excessive task length and difficulty.

As a preliminary estimate, I felt that 2 - 3 months of full time effort would be required to bring the project to fruitation. We agreed to contract the work on a per day basis, plus expenses. The complete job is to involve the following:

1. Bring experiment to point of being ready for data collection
   a. Design of experiment
      1. materials
      2. procedure
      3. statistical design
   b. Thorough piloting and pretesting
2. Keep diary explaining work done, problems, decisions taken
3. Maintain close communication with Andrew Singer, especially during design phase.
4. Running of experimental subjects
5. Statistical analysis
6. Preparation of publication-ready manuscript

July 19, 1979

Called Henry and we agreed that my initial responsibility is to bring the experiment into a ready-to-run state within 20 working days. This includes items 1 through 3 above.

July 23, 1979

Spent the day familiarizing myself with the experiment and drawing up a list of concerns and ideas. The following seem to be problems with the current design:

1. Experiment too lengthy and cumbersome.
2. Between-subjects design is not the best choice. Introduces too much variability in a situation where power to detect differences is an overriding consideration. The same subjects should use both editors. Order and practice effects can be handled with pretraining and counter-balancing.
3. "Natural language" is not spelled out as a model. This may be the experiments most important defect because it makes it impossible to specify exactly what is being tested. I hope to get a better sense of what the designers intended once I can use the editors myself.
4. Plans for data analysis are primitive and inadequate. We should plan to collect and analyse many variables, not just total time to completion. I agree completely with Andrew Singer's views on this (as represented in this dairy.)
5. The "natural language" editor contains a lot of unnatural stuff, for example, line prompts, slashes. The algebraic notation for describing the natural language editor violates the intent of the experiment, I believe. I propose to make the "natural language" editor, and the instructions more natural.
6. I don't believe the post-test is a valid measure since it is so far removed from conditions of actual use. I strongly favor bringing the subjects back for another editing session.

Clearly I need access to the existing pilot data, or else I must generate my own. I also need experience with the editors to get a feel for the experiment.

July 25, 1979

Visited COINS, logged in under Henry's account, and arranged for what appear to be the experimental files to be reinstated on the system. Searched Henry's office for experimental materials but found none. Acquired a number of systems manuals and studied them.

July 30, 1979

Continued review of experimental materials. Ideally, I would like to implement a suggestion discarded early on by the original investigators--intermix both types of commands in the same editor. Their objection (confounding of command type with grammar) could be easily met by counter-balancing commands across subjects. The experiment would be simple, quick, and powerful. I suspect though that it may be impractical to implement

The next best thing would be to simplify the manuals, editors, and texts enough so the both can be administered in one session. For one thing, I'm convinced that the materials are much too long and complex as it is so that most of the variability comes from difficulty in mastering these materials. Secondly, repeated measures is the only way to go, due to the power issue.

Worked on simplifying the editors and the manuals. It appears that a number of the more complex commands can be done away with. So far I have the instructions down to about two-thirds of their original length.

Formalized alternative experimental designs. All have in common the administration of a practice session followed by exposure to both editors, all in a single session. A follow-up test a week later would involve retesting using both editors.

Called Henry who will arrange for me to meet with a student whom we may hire to help with the experiment.

July 31, 1979

Visited Henry to explain ideas so far. He was generally receptive but felt that it was important to discuss certain issues in depth with Andrew, specifically:

1. What should be done with the post-test? Should it be an editing session? Should it be eliminated? What role does it serve in the experiment as a whole?
2. I recommend a repeated measures design with subjects exposed to both edi-ors during the same session. Are there any problems with this heavy mixing?
3. To what extent can or should the editors be shortened? Henry and I feel they can be simplified considerably.
4. What data are worthwhile to collect?

Later in day met with Richard Shire who answered our notice for programming and systems help. Explained the project to him. He seemed enthusiastic and component so I asked him to prepare print outs of Bill Seymour's old files.

August 1, 1979

Technical difficulties prevented getting print outs of the old files. Talked further with Rich whose initial reaction to the experiment has been that the natural language editor was not a very good one. It seems he is experienced with and prefers more powerful editors. His comments are interesting in light of subsequent realizations (see August 6, 1979). Asked Rich to get print outs of all files and to attempt a preliminary catagorization. Still unable to reach Andrew Singer.

Rea

August 6, 1979

Rich had left and catagorized a number of Seymour's files. Most of what we need to run the experiment seems to be there with the important exception of the editor source listings. Went back to the archives and discovered some likely candidates for these. A major problem is that the files that appear to pertain to the experiment are not annotated so we will be slowed considerably while we attempt to piece the original experimental package back together.

Reached Andrew and had a lengthy and wide-ranging discussion with him.

The most critical point to emerge is that we cannot proceed with the experiment under the assumption that it will eventually be received by an impartial, objective audience familiar with good research practices. That is, political considerations as well as scientific ones must enter into the design. The audience will be hostile and unreceptive. This sort of experiment is radical in the eyes of most programmers and systems people. The reason, we agreed (and incidently, the reason that human factors work generally has failed to impact software design) is that the idea of designing an effective man-machine system is in contradiction to the "man versus machine, man competes with machine, man attempts to outsmart system" approach common among dedicated computer users. For these individuals, interaction with computers is an end in itself, not a means to an end. The thrill and sense of power in getting the machine to do what you want it to becomes the important source of motivation rather than accomplishment of the task itself. A good case in point is the frequently repeated situation where a computer enthusiast develops programs to accomplish tasks that could have been done by hand with far less investment of time and effort.

So we come to the politics of experimental design. On these grounds, Andrew has reservations about the repeated measures design. Readers will immediately assume that experienced subjects are biased in favor of one or another editor and that this will affect their performance. The between subjects design is less open to this criticism since the purpose of the experiment will be less obvious to the participants.

I am unconvinced, on grounds of research methodology, that this is a problem, since I believe that most individuals, given proper instructions and incentives, will be motivated to perform well on tasks such as those proposed in this study. But this does not answer the political question.

If we use a between-subjects design, I believe that we enormously reduce our chances of showing anything at all. This is because of between-subjects variability which is always high on complex cognitive tasks and will be especially so on this one given the idiosyncratic and varied approaches that individuals have to editing. All of this variability reduces observed effect in a between-subjects experiment but it is factored out in a repeated measures design.

My current feeling is to measure biases directly by means of a questionnaire given prior to experience with the editors. (It only makes sense to do this with the experienced subjects).

Statistically, it is possible to use the bias measurements as a covariate. This means that the performance measures themselves would not be analysed but rather those measures corrected for any correlation with the bias scores. Thus any significant differences found between editors would be differences over and above any that could be accounted for by the bias scores.

Since the potential benefits of the repeated measures desing are great I propose to stick with it at least for pilot testing. I would consider a change in design if pilot testing shows a strong relation between bias and performance.

Another topic of discussion concerned the extent to which RS is really a natural language editor as well as what it meant by natural language. Andrew described the basic rationale of the experiment as follows: People have vast experience with natural language. Can we avail ourselves of this experience when attempting to design easyt-to-use systems?

RS clearly does not have many natural language features. It does have common words and phrase structure. I pointed out that this may not be the critical aspect of natural language from the human factors perspective. My guess is that ability to use synonyms and multiple syntactic constructions is more important. Apparently, Andrew had argued this point in other contexts.

As to the specific issues mentioned under July 31, 1979, Andrew felt that:

1. The post-test in its present form should be eliminated. If long term retention is an experimental objective then a second editing session should be included. Otherwise, there should be no post-test at all.
2. See above.
3. The experiment is too long and cumbersome. However, the editors are probably "minimal" already in terms of what professionals would consider adequate. Further reductions might destroy the credibility of the experiment.
4. As many variables as possible should be collected and analysed.

Andrew agreed that the algebraic notation should be eliminated from the manuals. Also, both of use felt that the text was too interesting and that the required editings seemed arbitrarily chosen. Better to have a more neutral text--also one in which the required changes are repetitious--this will give subjects a chance to develop stable strategies.

August 6, 1979

Located and restored original files allowing us to run original experiment. Henry and I agreed: (1) to eliminate only a few editor features, and (2) we will use the repeated measures design with a bias questionnaire, and (3) the text to be edited will be revised along lines discussed on August 6, 1979, and (4) the post-test will be eliminated.

August 8 through August 16, 1979

Worked on revising materials. Selected company financial statements for textual material. Also established a more precise conceptual basis for the experiment. On August 16, 1979, gave Henry a complete set of experimental materials.

August 21, 1979

Found pilot data from original experiment. Reinforced my conviction that materials were too complex. Subjects understood the editors poorly. Also dependent measures incapable of showing differences (e.g. whether or not task was completed is only a binary variable--no resolution.)

August 22 through August 30, 1979

Prepared further revisions of the command summaries and manuals--difficult to prepare these well. Finished materials on August 30, 1979.

September 4, 1979

First dry run of a subject--only minor problems.

September 5, 1979

Realized problems with efficiency index--revised it to reflect not individual commands but rather the number of successful text changes. Efficiency is now defined as the number of successful changes divided by the total number of commands issued. Ran first pilot subject today with such successful results that we will start the actual experiment. Simplified the dependent measures--will only look at the percentage of the task completed, the percentage of erroneous commands, and the editing efficiency--and, in addition, the preference data.

September 13, 1979

The experiment began today but problems with system load restrict running times to early morning and evening hours.

October 12, 1979

Ran last subject today.

October 15, 1979

Some reflections. The involvement of an experimenter to help the users learn the editors was essential to the success of the experiment. It made it possible for the inexperienced users to gain a reasonable command of the editors in a short period of time. The attempt of the original experimenters to present the users with only written materials was a mistake--no matter how well prepared the materials, subjects will misunderstand them and take forever to learn them.

Presenting the editors with a command summary table proved very effective--in fact, almost no one needed to consult the detailed manuals.

Individuals have highly individual strategies of editing--but ones that are consistent unto themselves. This means that using a repeated measures design was the right choice--individual variability is just too high to obtain results, otherwise, unless many, many subjects were run. Editing strategy would make a very interesting future study.

Many of the subjects seemed to feel that only the written changes need be made to the text--that correct spacing and alignment was not important. Othere were concerned with spacing. Again, the repeated measures design insures that this variability does not affect the comparison of editors. In scoring, I treated making an indicated substitution separately from getting the spacing in the tables correct.

All of the subjects enjoyed participating in the experiment. Average time involved was 1 1/2 hours. Motivation and interest were high.

REFERENCES

[Boies 1974]

Stephen J. Boies
User Behavior on an Interactive Computer System
IBM SYSTEMS JOURNAL, Number 1, 1974, p. 2-18

[Cuandra 1971]

Carlow A. Cuandra
On-Line Systems" Promise and Pitfalls
JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE, March-April, 1971

[Epstein and Arlinsky 1965]

W. Epstein and M. Arlinsky
The Interaction of Syntactical Structure and Learning Instructions.
PSYCHOLOGICAL SCIENCE 3, 1965

[Holt and Stevenson 1977]

H.O. Holt, and F.L. Stevenson
Human Performance Considerations in Complex Systems
SCIENCE, Volume 195, 1977, p. 1205-1209, 1977

[Ledgard and Singer 1977]

Henry Ledgard and Andrew Singer
The Case for Human Engineering
Computer and Information Science Department Technical Report #77-11, University of Massachusetts, Amherst, 1977

[Myers 1966]

Jerome L. Myers
FUNDAMENTALS OF EXPERIMENTAL DESIGN
Allyn and Bacon, Boston, MA, 1966

[Seymour 1978]

William Seymour
Diary of a Human Factors Experiment
Computer and Information Science Technical Report 77-14
University of Massachusetts, Amherst, 1978

[Singer et al. 1978]

Andrew Singer, Henry Ledgard, and Jon Hueras
The Annotated Assistant
Computer and Information Science Technical Report
University of Massachusetts, Amherst, 1978

[Weist and Dolezal 1972]

R. Weist and J. Dolezal
The Effect of Violating Phrase Structure Rules and Selectional Restrictions on TEP Patterns
PSYCHOLOGICAL SCIENCE, Volume 27(6), p. 355-356, 1972