The Evaluation of Text Editors: Methodology and Empirical Results

Teresa L. Roberts *Xerox Office Systems Division*

Thomas P. Moran *Xerox Palo Alto Research Center*

Thomas J. Moran's major research interests are in mental models of systems, the learning of systems, and the nature of expertise in using systems, as well as the formalization of the issues and processes of designing systems. Teresa L. Roberts received her Ph.D. in Computer Science from Stanford University in 1979.

Authors' Present Addresses: Teresa Roberts, Xerox Office Systems Division, 3333 Coyote Hill Road, Palo Alto, CA 94304 Arpanet Roberts. PA @ PARC-MAXC; Thomas Moran, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304 Arpanet Moran. PA @ PARC-MAXC.

0001-0782/83/0400-0265 75¢.

***ABSTRACT:** This paper presents a methodology for evaluating text editors on several dimensions: the time it takes experts to perform basic editing tasks, the time experts spend making and correcting errors, the rate at which novices learn to perform basic editing tasks, and the functionality of editors over more complex tasks. Time, errors, and learning are measured experimentally; functionality is measured analytically; time is also calculated analytically. The methodology has thus far been used to evaluate nine diverse text editors, producing an initial database of performance results. The database is used to tell us not only about the editors but also about the users—the magnitude of individual differences and the factors affecting novice learning.*

1. INTRODUCTION

Text editors are the most heavily used programs on interactive computing systems since the advent of time-sharing systems (e.g., [1]). Text editing, or word processing, is also a very pervasive use of personal computers [15]. There are probably hundreds of different text editors in use today: many computation centers have their own local editors, and new computers often come with their own text editors. System programmers cannot seem to resist the temptation to design a better text editor. Heated debates rage over computer networks about text editor design. Yet, remarkably little objective information is known about the relative advantages of different kinds of editing paradigms.

Systematic study of text editors is hampered, at least partially, by the complex of issues surrounding text editor usage. Text editors are flexible tools that are used for a wide variety of purposes, since many kinds of human communication are done by text. Simple informal notes, letters and memoranda, structured text (such as lists and tables), reports and specifications (requiring sophisticated formatting and layout), and program code (structured differently from narrative text) are all applications for which text editors are regularly used. There are many different kinds of editor users—first-time novices, hardened experts, occasional users, and users with specialized applications that lead them to know how to perform some tasks well and other tasks not at all. Finally, there are many different measures of the quality of user-editor interaction, including both objective measures of performance, such as time and errors, and subjective measures of acceptability, such as feelings of enjoyment, clumsiness, and so forth.

The study of text editors up to now has been dominated by functional descriptions of editors, both by proponents of particular systems (e.g., [16]) and by neutral

evaluators (e.g., [10, 14, 8]). These reports mainly present subjective opinions as the basis for comparing different systems, either by deciding ***a priori*** what features are desirable or by informally trying out the systems to get a feel for what works well and what is lacking. Various arguments, which on the surface seem reasonable, are also used to defend the conclusions in these reports, but the validity of these arguments is seldom tested. The purpose of the present study is to obtain objective, replicable results. A survey of related behavioral studies done up to this time is given in [6].

Our purpose in this paper is to present a ***standardized evaluation*** of text editors. This kind of evaluation may be contrasted with a ***specific evaluation***, which is tailored to a particular purpose or situation, such as the evaluation of a set of editors to determine their utility in a particular working environment. A standardized evaluation does not make assumptions about the particulars of any given situation, nor does it cover all of the various aspects of editor usage. It focuses on the common properties of text editors rather than on the idiosyncracies of particular editors. A standardized evaluation attempts to address the most fundamental issues and is thus applicable to a variety of editors. A familiar example of a standardized evaluation is the EPA rating of automobile gasoline mileage. While the conditions used to obtain the EPA rating do not match the driving conditions of any specific car, the ratings do relate to common driving situations. Thus, the ratings can be used to compare different cars and, to some extent, can be adjusted to tell about specific driving situations.

A benefit of using a standardized evaluation over a period of time is the accumulation of a database of consistent information about editors. This gives a standard for interpreting the results of any new evaluation, a critical factor missing from many specific evaluations (e.g., [7]). One of our goals in proposing a standarized evaluation is to initialize a database of information about the population of existing editors.

The methodology we present here evaluates computer text editors from the viewpoint of the performance of their users—from novices learning the editor for the first time to dedicated experts who have mastered the editor. Objectivity, thoroughness, and ease-of-use were the criteria used in creating this methodology. ***Objectivity*** implies that the methodology not be biased in favor of any particular editor's conceptual structure. ***Thoroughness*** implies that multiple aspects of editor usage be considered. The methodology focuses on four dimensions of editor usage that are behaviorally fundamental and practically important.

The ***Time*** to perform basic editing tasks by experts.
The ***Error*** cost for experts.
The ***Learning*** of basic editing tasks by novices.
The ***Functionality*** over a wide range of editing tasks.

Ease-of-use means that the methodology should be usable by editor designers, managers of word processing centers, or other nonpsychologists who need this kind of evaluative information, but who have limited time and equipment resources.

The structure of this paper is as follows: In Sec. 2, we describe the evaluation methodology. In Sec. 3, we apply the methodology to nine different text editors, presenting and discussing the empirical results, and assessing the methodology itself. In Sec. 4, we turn the empirical results around to gain some insight into user performance with computers, particularly in the areas of individual differences and novice learning.

2. DESCRIPTION OF THE METHODOLOGY

The methodology is based on the specific kinds of tasks involved in text editing. It consists of experimentally measuring user performance on three dimensions—Time, Error, and Learning—and on an analysis of Functionality. Also, expert performance time can be calculated analytically.

2.1 Taxonomy of Editing Tasks

An evaluation scheme for editors needs to have a common ground on which to compare different kinds of editors. Editor design features (e.g., "modeless" insertion of new text vs. having an "insert mode") and design concepts (e.g., table creation using sequential text with formatting characters such as tabs vs. using a two-dimensional structure) cannot serve this role, since the features and concepts differ so much from editor to editor. There is no evidence that one feature is always better than another. In fact, the overall consistency in how well the different design features of the editor fit together may well be more important than any individual feature in determining the quality of the editor.

What is constant across all text editors, in contrast to design features, is the editing tasks they permit their users to accomplish. Thus, the methodology here is based on a taxonomy of 212 editing tasks that can potentially be performed by a text editor. These tasks are specified in terms of their effect on a text document, independent of any specific editor's conceptual model [9]. The organization of the task taxonomy, along with a sample of tasks in each category, is given in Figure 1. The Functionality dimension of an editor is measured with respect to the set of tasks in this taxonomy, by assessing how many of the tasks the editor can perform.

Comparisons between editors on the performance dimensions (Time, Error, and Learning) must be based on tasks that all editors can perform. For this purpose, we identify a small set of ***core editing tasks*** (see Figure 2). The core tasks are the ones that all text editors, by definition, can perform; they are also the most common editing tasks in normal text-editing applications. Most of the core tasks are generated by applying basic text editing operations (e.g., ***insert, delete, replace***) to basic text entities (e.g., ***characters, words, lines***). Also included in the core set are the tasks of accessing and saving documents and the simplest text-display and text-addressing operations.

A lengthy specification is required to instruct an evaluator to carry out this methodology. In this paper we can give only enough information to make clear the basic structure and procedure of the methodology and the resulting measures. Full instructions and materials for running the evaluation tests and analyses may be found in the report by Roberts [11].

2.2 The Time Dimension

The time it takes expert users to accomplish routine text modifications is measured by observing expert users as they perform a set of ***benchmark tasks***, which are drawn from the core tasks.

Benchmark. There are 53 editing tasks in the benchmark, embedded in four documents: a short interoffice memo, two two-page reports, and a six-page chapter from

FIGURE 1. Taxonomy of Editing Tasks on which the Evaluation Methodology is Based.

Modify Document
- **Content and structure of text**
  - Characters, words, numbers, sentences , paragraphs, lines, sections, document
  - References [e.g., keep up-to-date references to section numbers in the document]
  - Sources for text or attributes [e.g., make the text layout be the same as in another document]
- **Layout of running text and structure**
  - Inside paragraphs [e.g., indent the first line of a paragraph so far from the left margin]
  - Headings, random lines [e.g., center]
  - Interparagraph layout [e.g., leave so much space between paragraphs]
  - General [e.g., lay out document in so many columns]
- **Page layout**
  - Every page [e.g., print a page heading that includes the current section number]
  - Non-mainline text [e.g., position footnotes at the bottom of the page]
- **Attributes of characters**
  - Line break [e.g., automatic hyphenation]
  - Shape [e.g., boldface]
- **Tables**
  - Column beginning [e.g., columns are equally spaced]
  - General alignment [e.g., align the column on the decimal points]
  - Modify alignment [e.g., swap the positions of two columns]
  - Treatment of table entries [e.g., line up the left and right edges of (justify) each table entry]
- Summary of text [e.g., table of contents]
- Special applications [e.g., mathematical formulas]

Locate Change (Addressing)
- Text [e.g., find text which has specified content]
- Structure [e.g., find the next section heading]
- Layout/Attributes [e.g., find a boldface character]
- Misc.

Program Edits (Control)
- Command sequences [e.g., invoke a sequence of commands with parameters]
- Control structure [e.g., repeat a sequence of commands a specified number of times]
- Tests [e.g., compare strings for alphabetical order]
- Storage [e.g., store pointers to places in documents]
- User control [e.g., ask user for parameters during execution]
- Preexisting composite commands [e.g., sort a sequence of text strings]

Find Task or Verify Change (Display)
- Display text and layout [e.g., show the outline structure of the text]
- Display system state [e.g., show where the selection is relative to the whole document]

Miscellaneous
- Hardcopy
  - Draft copy [e.g., print with extra space between lines]
  - Misc. [e.g., print on envelopes]
- Intermediate Input/Output [e.g., save away the current version of a document]
- Other [e.g., perform arithmetic on numbers in the document]

a philosophy book. The types of tasks in the benchmark are randomly drawn from the core tasks, and the locations and complexities of the benchmark tasks are also randomly distributed. The distribution of tasks in the benchmark is more uniformly distributed than one would observe in normal text-editing work, the benchmark giving more emphasis to the more complex kinds of tasks (most real-world editing tasks are simple text modifications involving a small number of characters). For example, tasks involving "tricky" boundary conditions are over-represented in order to identify special cases, such as insertion at the beginning of a paragraph, which an editor may treat awkwardly. The benchmark also underrepresents the typing of lengthy new text, since such typing performance is more a reflection of the skill of the user than of the quality of the editor. We will discuss later how to relate this benchmark to other distributions of tasks.

Subjects. Four expert users were tested individually on the benchmark. The evaluator should select the set of subjects to represent the diversity of the expert user community: at least one user should be ***nontechnical*** (i.e., with no programming background) and at least one should be ***technical*** (i.e., know how to program). Four is the absolute minimum number of subjects needed to get any reliability of measurement and to get some indication of individual user variation.

FIGURE 2. Core Editing Tasks used in the Methodology for Expert Time Performance and Novice Learning.

Core tasks consist mainly of the cross-product (except for a few obvious semantic anomalies) of the following basic editing operations applied to the following basic text objects:

Operations:	Objects:
insert	*character*
delete	*word*
replace	*line*
move	*sentence*
copy	*paragraph*
transpose	*section*
split	
merge	

For example:

—*insert character(s)*
—*insert word(s)*
—*delete character(s)*
etc.

Core tasks also contain the following miscellaneous tasks:

—*display a continuous chunk of text*
—*address a specified place*
—*address according to content*
—*make a document available for editing*
—*put a document away*
—*start a new document*

Note. The formal definition of the core in the task taxonomy also includes operations on the object *number*; however, no tasks using numbers were included in the experiments. The learning experiments omitted the operation *transpose* since it can be regarded as an optimization of two *moves*.

Measurement. The evaluator measures the performance in the test sessions with a clock and a stopwatch, measuring the overall performance time with the clock and the times spent in error with the stopwatch. The evaluator also notes whether or not each task is performed correctly. When the subject is finished with the tasks, the evaluator asks the subject to make a second pass to complete any incorrectly done tasks. This relatively crude method of measurement is used because it is easy for anyone to run (not everyone has an instrumented editor or a videotape setup, but anyone can acquire a stopwatch) and because stopwatch accuracy is sufficient.[1]

Error-free and Error Time. The benchmark typically takes about 30 minutes of steady work to complete. The elapsed time in the experiment is partitioned into error-free time and error time, according to two types of observed behavior. The ***error time*** is the time the user spends dealing with errors (see below for more detail), and the ***error-free time*** is the elapsed time minus the error time.

Scoring. The individual user's Time score is the average error-free time to perform each task (i.e., the total error-free time divided by the number of tasks). The overall Time score is the average score for the four subjects.

[1] The reliability of the measurements is determined more by the small number of subjects than by the accuracy of measurement.

2.3. The Error Dimension

The effect of errors in an editor is measured by the ***error time***, which is the time cost of errors on the benchmark tasks. The course of a typical error includes committing the error, discovering it, correcting it, and then resuming productive behavior. Error time consists of all the activity up to the resumption of productive activity [4]. Only those errors that take more than about 15 seconds to correct are counted by the evaluator (which is the best that can be done with a stopwatch). Thus, the time for typographical and other simple errors is not included in the error time. We do not know exactly how close this method approximates the true error time, but the true error time is not likely to be dominated by the time in these small errors. In addition to the time for the immediately corrected errors, the time for the second-pass corrections is also counted in the error time.

Scoring. The individual Error score for each user is the user's error time expressed as a percentage of his/her error-free Time score.[2] The overall Error score is the average score for the four expert users.

2.4 The Learning Dimension

The ease of learning of an editor is tested by actually teaching four novice subjects, individually, to perform the core editing tasks.

Subjects. Each subject must be a ***novice*** to computers (defined as someone with ***no*** previous experience with computers or word processors). This gives us an easily defined baseline measure of learning, that is, from zero experience.[3]

Teaching Paradigm. The learning tests are performed in a one-on-one, oral teaching paradigm, with an instructor individually teaching each novice the editor. Although more expensive than group-teaching or self-teaching paradigms, this paradigm has the crucial advantage that it is adaptable to the individual learner. The other paradigms are more rigid and may tend to magnify the differences between different learners, which obscures the learnability of the editor itself. For example, in a self-teaching paradigm using the editor's documentation, a learner can easily get confused on a point because of a short lapse of attention or because of the particular wording of the documentation and not because the point is inherently difficult. In the one-on-one paradigm, on the other hand, the instructor can respond to the particular difficulties of each learner by explaining things in a different way, by correcting misconceptions, and so forth.

Teaching Procedure. The teaching procedure is structured as a series of five instruction–quiz cycles. In each cycle, the instructor first instructs the learner on some new tasks or corrects the learner's difficulties, and during this time the learner is allowed to practice performing tasks on the system; finally, the learner is given a quiz to test what tasks s/he can do independently. The learner paces the session, deciding how much to practice, when to take the quiz, and so on.

The methodology includes a standard syllabus specify-

[2] Thus, the total time to perform an average benchmark task is $T + Te$, where T is the error-free Time score and e is the Error score.

[3] More and more people today have some exposure to computers, and it may become more important to look at the learning users experienced in other systems. However, this would present difficult methodological problems in assessing their degree of experience and the similarity of their experience to the editor to be learned.

ing what core tasks are to be taught on each cycle. However, it is up to the instructor to determine which specific editor commands and facilities to teach in order for the subject to be able to accomplish the core tasks. The structure of a particular editor might also make it necessary to slightly alter which tasks are taught in which cycle. The teaching procedure is strongly method-oriented; by "teaching tasks" we mean teaching methods to accomplish the tasks.

The quizzes consist of documents marked with changes to be made (similar to the benchmark performed by the expert users). Only a sample of the core tasks appears on each quiz. Not all tasks on a quiz have necessarily been taught up to that point, which allows learners to figure out, if possible, how to do tasks that have not been explicity taught. During the quizzes, the learners are given access to a one-page summary sheet listing all the editor commands taught. Thus, a learner is not hung up a long time on a quiz because of a simple difficulty, such as not being able to remember the name of a particular command.

Scoring. The amount that a subject learns is measured by counting the number of different task types the subject is able to perform on the quizzes. Only half-credit was given if the subject performed a task incompletely or had to look at the summary sheet. The individual Learning score is the amount of time taken for the learning session divided by the total number of tasks learned, that is, the average time it takes to learn how to do a task. The overall Learning score is the average Learning score for the four novice learners.

2.5 The Functionality Dimension

The range of functionality available in an editor is measured by analyzing the editor against a checklist of tasks covering the full task taxonomy (Figure 1).

The Analyst. The editor is rated on the functionality checklist by a very experienced user of the editor, the ***analyst***, who uses whatever documentation material is necessary to ensure accuracy.

Rating Criteria. Rating the functionality of an editor on a task involves deciding whether the task can or cannot be done with the editor. This is not a simple binary decision. Almost any task can be performed on almost any editor with enough effort. Consequently, the editor is given full credit for a task only if the task can be done efficiently with the editor. It is given half-credit if the task can be done awkwardly, which can appear in several guises: repetition of commands, excessive typing of text, limitations in parameter values to the task, interference with other functions, substantial planning required of the user, etc. The editor is given no credit for a task if it cannot be done at all (such as trying to specify an italic typeface on a system designed for a line printer) or if doing the task requires as much effort as retyping all the affected text (such as having to manually insert page numbers on every page).

Scoring. The overall Functionality score is the percentage of the total number of tasks in the task taxonomy that the editor can do, according to the rating criteria. This score may be broken down into subscores according to the classes of tasks in the taxonomy, to show the strengths and weakness of the editor.

2.6 Calculation of Expert Performance Time

The error-free performance time of an expert using an editor can be calculated analytically, using the Keystroke-Level Model [3, 4]. This model predicts expert performance time by counting the number of physical and mental operations required to perform a task and by assigning a standard time for each operation. The model counts operations at the grain-level of keystrokes: typing, pointing at a location on the display with a pointing device, homing the hands onto a device, mentally preparing for a group of physical operations, and waiting for system responses. The Keystroke-Level Model analysis gives a precise characterization of methods for accomplishing tasks.

When the model is applied to the set of benchmark tasks, it produces a calculated performance time for a "standard expert" that can be compared to the experimentally measured times. However, making this calculation requires the evaluator to predict what methods an expert user would use to perform the benchmark tasks, since the model requires that the methods be specified as input. In the absence of knowledge about the style of expert user interaction, the most useful heuristic is to first identify the common, frequently used commands of the editor and to pick the optimal method for each task within that set of commands. The fact that the experimental subjects sometimes use methods different from those predicted, plus other differences between the assumptions of the Keystroke-Level Model and the test conditions in this methodology (e.g., the inclusion of small errors) leads us to expect small-to-moderate differences between the calculated performance and the experimental results.

3. EVALUATION OF NINE TEXT EDITORS

Nine text editors have been evaluated using this methodology, both as a test of the methodology and for the inherent interest in the results. The results of these evaluations provide the beginnings of a database of empirical results giving us behavioral data on user performance, as well as the basis for comparing editors.

3.1 Description of the Editors

The nine text editors evaluated are: TECO [20], WYLBUR [24], EMACS [23], NLS [18, 19], BRAVOX [21], BRAVO [22], a WANG word processor [26], STAR [27], and GYPSY [25]. These represent a wide variety of text editors and word processors, some in wide use around the country and some experimental. The first two of these editors were designed for teletypelike terminals, and the rest were designed for display-based terminals or personal computers. The intended users of these editors range from devoted system hackers to publishers and secretaries who have had little or no contact with computers.

Text editors are complex interactive systems. Thus, it is difficult to succinctly describe the design of these nine editors. Figure 3 attempts to characterize the editors according to a set of commonly discussed design features. For example, the Command Invocation column describes the design feature concerned with the ways in which a user designates commands to the system. The nine editors cover a wide range of choices for this feature: (1) type all or part of an English verb, (2) type a one-letter mnemonic for the command name, (3) hold down a control key while typing a one-letter mnemonic, (4) type a one-letter mnemonic on a chordset, (5) press a special function key, (6) select a command from a menu on the display.

FIGURE 3. Feature Description of Nine Text Editors.

Editor [Ref.]	Feature								
	Display	Auto Line Wrap[a]	Strong Line Concept[b]	Text Units	Command Invocation	Insert Mode	Means of Addressing[c]	Addressing Hardware	Computer Processor[d]
TECO [20]	TTY[e] style	No	Yes	Characters, lines	1-letter mnemonic	Yes	Relative to current position	Keyboard	PDP-10 equivalent, via 3Mb net
WYLBUR [24]	TTY[e] style	No	Yes	Characters, lines	English-like, abbreviated	Yes	Absolute line numbers	Keyboard	IBM 370, 1200 baud
EMACS [23]	Partial page	Yes	Yes	Characters, words, lines, sentences, paragraphs	1-letter mnemonic, control keys	No	Relative to current position	Keyboard	PDP-10 equivalent, approximately 1200 baud
NLS [18, 19]	Partial page	Yes	No	Characters, words, paragraphs	1-letter English-like on keyboard or 5-key chordset	Yes	Screen position	Mouse	PDP-10 with local processor
BRAVOX [21]	Full page	Yes	No	Characters, words, lines, paragraphs	1-letter mnemonic, menu, function keys	No	Screen position	Mouse	Xerox Alto personal computer
BRAVO [22]	Partial page	Yes	No	Characters, words, lines, paragraphs	1-letter mnemonic	Yes	Screen position	Mouse	Xerox Alto personal computer
WANG [26]	Partial page	Yes	No	Characters	Function keys	Yes	Screen position	Step keys[f]	Stand-alone Wang word processor
STAR [27]	Full page	Yes	No	Characters, words, sentences, paragraphs	Function keys, menus	No	Screen position	Mouse	Xerox 8000 processor
GYPSY [25]	Partial page	Yes	No	Characters, words, paragraphs	Function keys	No	Screen position	Mouse	Xerox Alto personal computer

[a] Automatic line wrap means that during type-in a new line is automatically begun when a word overflows the old line, without any intervention from the user.

[b] This refers to editors that require the user to type RETURN at the end of each line of text. Usually, this also means that there is an explicit CARRIAGE-RETURN character at the end of each line in the internal representation of the document.

[c] This refers to the primary means of addressing (all editors have the ability to search).

[d] Time-sharing computers were used under conditions of light load. Terminals and computer displays were all CRTs, except that one WYLBUR user preferred her own hardcopy terminal.

[e] A TTY (teletype) style display is one that does not continuously show the state of the document, but only shows the sequence of commands entered by the user. Snapshots of pieces of the document are displayed when the user explicitly asks for them.

[f] Four keys with arrows on them, which move the cursor up, down, left, and right (see [2]).

Figure 3 also gives the conditions under which the editors were used for the experiments. For example, TECO was run on a time-sharing machine connected to a terminal over a 3-megabit local network, while the WANG word processor was run on its own stand-alone hardware. Note that the methodology does not provide an evaluation of an editor in the abstract, but only of a particular implementation under a particular set of conditions. It is possible that the particular conditions (e.g., the quality of the terminal or the bandwidth of its connection to the central processor) dominate the abstract characteristics of the editor (e.g., its command language conventions) in determining an expert's performance. Therefore, an attempt was made to run each editor under reasonably optimal conditions, in order to make the overall evaluation results as generally useful as possible.[4]

Figure 4 gives a different characterization of the editors. It shows in detail how a user would go about performing a specific word-replacement task in each of the editors, using the notation of the Keystroke-Level Model (the footnote to the figure lists the different types of Keystroke-Level Model operations). For example, it can be seen that the editors described in Figure 3 as having an insert mode (TECO, WYLBUR, NLS, BRAVO, and WANG) all require the typing of a special character (preceded by a mental operation) after the insertion to terminate the insertion of new text. On the other hand, the "modeless" editors (EMACS, BRAVOX, STAR, and GYPSY) do not require any operations after typing in new text. These methods also show where moving the hands from the keyboard to the pointing device and back (homing) add extra motions to the methods used with editors which have a mouse or step keys (NLS, BRAVOX, BRAVO, WANG, STAR, and GYPSY).

This Keystroke-Level Model analysis can be used to calculate the expected expert performance time for each editor, and to give a detailed quantitative decomposition of the times for each type of operation in each editor. To do this, the Keystroke-Level Model analysis was applied to all the benchmark tasks for each of the nine editors. The calculated task times thus obtained were averaged over the 53 benchmark tasks to give times for an "average editing task" for each editor. Figure 5 presents these first empirical (not experimental) results.

Figure 5(a) gives the calculated average task times for each editor. This leads us to expect a certain pattern of experimental results, for example, for there to be an overall factor of 2.5 between the fastest and the slowest editors. The figure also shows how each average task time is decomposed into the times for each operator type. For instance, the cost of slow system response stands out clearly. If EMACS had been run on a fast terminal, its speed would be faster than NLS's; STAR would be the fastest editor of all if system response times for all editors could be effectively reduced to zero. A weak point of the WANG, on the other hand, is the pointing time required by the step keys; it would be improved at least 2 sec/task (over 10 percent) by using a mouse.

The task time decomposition can also be considered as a percentage of total task time, as shown in Figure 5(b). This shows, for example, that homing time between the keyboard and pointing device is not a major problem (except perhaps with the WANG, which relies heavily on function keys that are separate from the main typing array). An interesting contrast exists between TECO and WYLBUR. Both use the same set of operations: Acquire, Keying, Mental time, and system Response. But TECO, with its emphasis on minimal typing, only spends one-third of its user's time in typing, while WYLBUR spends over half. This is paid for, however, in Mental time, where the ratios are reversed.

3.2. Overall Evaluation Results

All nine editors were run through all the evaluation tests. According to the methodology, the overall evaluation of a text editor is a four-tuple of numbers, one numeric score from each dimension. The overall evaluation scores for the nine editors are presented in Figure 6.

Differences were found between the editors on all dimensions. The expert Time results show, for instance, that TECO, WYLBUR, and EMACS are the slowest editors and that GYPSY and STAR are the fastest. Most of the display-based systems are about twice as fast to use as the non-display systems. The difference between the fastest and slowest system was a factor of 2.5, as the Keystroke-Level Model analysis led us to expect. The Error dimension shows a range of a factor of 5 in the cost of errors between systems. On the Learning dimension, TECO is clearly the slowest to learn, with the next system being a factor of 2 faster to learn, and the rest of the editors ranging over another factor of 2 in learning speed. We also see large differences in the Functionality dimension, with scores ranging smoothly from under 40 percent of the tasks to almost 80 percent.

We see that no editor is superior on all dimensions, indicating that tradeoffs must be made in deciding which editor is most appropriate for a given situation. For example, consider the editor BRAVOX, which was developed at Xerox as an extension to the earlier editor BRAVO. Its purpose was to increase functionality and speed and to try out fashionable design features such as command menus and modeless text insertion. Is BRAVOX really an improvement over BRAVO? From Figure 6 we see that BRAVOX is indeed an improvement over BRAVO in Functionality; it is also faster to learn, possibly justifying the design innovations that were incorporated. The analysis in Figure 5 shows that BRAVOX should be faster than BRAVO, but that improvement does not materialize in the experimental Time score (the reason for this is unknown).

Reliability.[5] Thus far we have only been considering the mean evaluation scores for each editor without considering the variability associated with these scores. Figure 6 expresses the variability of each experimentally measured score by the Coefficient of Variation (*CV*),[6] which represents the between-user variability. We see that the variability is very high for the Error scores, but quite moderate for the Time and Learning scores. However, the statistical reliability of the scores depends on the number of subjects as well as on the variability. Since we ran only four subjects, only large differences between scores are statistically reliable. For example, we can say that WYLBUR is reliably faster to learn than TECO, but we cannot say

[4] EMACS was the only system for which optimal conditions were not found. The workstation used was actually a personal computer running a rather slow terminal-emulation package. This cut the effective communication rate between the main computer and the workstation to around 1200 baud, which is much slower than is often available with EMACS.

[5] In this paper we use the term "reliability" instead of the more usual term "(statistical) significance," since we are trying to emphasize the difference between *statistical* and *substantive* significance, the latter of which we call "importance."

[6] We use the *CV*, which is the Standard Deviation normalized by the Mean, instead of the Standard Deviation, because CVs are more constant across the different scores. That is to say, the absolute size of the variation is approximately proportional to the mean.

FIGURE 4. Example of the Use of each Text Editor: An Illustrative Method for Accomplishing the Specific Task of Replacing the Word "European" with the Words "Far Eastern".

Editor	Method (informal)	Method (Keystroke-Level Model encoding)[a]
TECO	Get task.	A[task]
	Place pointer after old word.	MK[s] 9K[European] MK[ESC]
	Delete previous 8 characters.	M 3K[-8d]
	Insert new words.	MK[i] 13K[Far Eastern] MK[ESC]
	Display line to verify.	M 2K[v ESC] R(0.4)
WYLBUR	Get task.	A[task]
	Get number of line with old word (system returns line 11).	M 2K[L␣] 10K['European] M 3K[' RETURN] R(1.0)
	Change old word to new words.	M 3K[ch␣] 10K['European] M 6K['␣to␣] 14K['Far Eastern] M 6K['␣in␣] 2K[11] MK[RETURN]
EMACS	Get task and find it on display.	A[task] S[European]
	Place pointer in old word.	M 4K[CTRL S e u] R(2.0)
	Back up to beginning of word.	2K[META B]
	Call Delete Word command.	M 2K[META D]
	Type new words.	13K[Far Eastern]
NLS	Get task and find it on display.	A[task] S[European]
	Call Replace Word command.	H[chordset and mouse] MK[r] MK[w]
	Point to old word.	P[European] K[OK] H[keyboard]
	Type new words.	13K[Far Eastern] MK[OK]
BRAVOX	Get task and find it on display.	A[task] S[European]
	Point to old word.	H[mouse] P[European] K[BUTTON2] H[keyboard]
	Delete old word.	MK[DEL]
	Type new words.	13K[Far Eastern]
BRAVO	Get task and find it on display.	A[task] S[European]
	Point to old word.	H[mouse] P[European] K[BUTTON2] H[keyboard]
	Call Replace command.	MK[r]
	Type new words.	13K[Far Eastern] MK[ESC] R(2.7)
WANG	Get task and find it on display.	A[task] S[European]
	Call Replace command.	H[function keys and step keys] MK[REPLACE]
	Select ends of old word.	P_S[E] K[EXECUTE] P_S[n] K[EXECUTE] H[keyboard]
	Type new words.	13K[Far Eastern] H[function keys] MK[EXECUTE]
STAR	Get task and find it on display.	A[task] S[European]
	Point to old word.	H[mouse and function keys] P[European] 2K[SELECT SELECT]
	Delete old word.	MK[DELETE]
	Type new words.	H[keyboard] 14K[Far Eastern␣]
GYPSY	Get task and find it on display.	A[task] S[European]
	Point to ends of old word.	H[mouse] P[E] K[BUTTON1] P[n] K[BUTTON1] H[keyboard]
	Type new words.	13K[Far Eastern]

[a] Methods are encoded in the Keystroke-Level Model [3] as a sequence of primitive operations that the user must perform. All operations are encoded as one of the following types of operations:

A	Acquire a task by looking at the manuscript (1.8 sec).
S	Search the display for the location of the task (2.2 sec).
K	Type a key or press a button (measured by typing tests; .23 sec used here).
P	Point to a location with a mouse (1.1 sec).
P_S	Point to a location with step keys (2.3 sec).
H	Home the hands on a physical device (.4 sec).
M	Mentally prepare for physical actions (1.35 sec).
R(*n*)	Wait *n* seconds for a system response (measured for each system).

The notation in square brackets after each operation is an informal comment telling, e.g., what keys are pressed. All operations, except A, S, and P_S, are the same as in [3]. The A and S operations used here simply encode [3]'s notion of task acquisition into new operations. P_S represents a type of pointing not covered in [3]. The time attributed to P_S comes from [2].

that it is reliably faster to use.[7] We also see that the Learning difference noted above between BRAVOX and BRAVO is reliable,[8] but the Time difference in the other direction is not. None of the differences in the Error dimension are reliable, because the between-subject variation is so high.

The reliability of the scores can be improved by increasing the number of subjects tested.[9] For example, consider the Time difference between WANG and STAR. Al-

[7] Quantitative formulas for computing which differences between scores are reliable (derived from the standard statistical concept of confidence limits) are given in the notes to Figure 6.

[8] This result was obtained using the actual variances of the BRAVOX and BRAVO data, rather than by using the general formulas given in Figure 6.

[9] Reliability, as measured by the confidence interval around a score, is approximately inversely proportional to the square root of the number of subjects used to determine the score.

FIGURE 5. Decomposition of the Calculated Editing Times into the Different Types of Keystroke-Level Model Operations.

(a) Average time (in seconds) per core editing task in each type of operation.

Editor	Operation Type								Total
	A	S	K	P	P_S	H	M	R	Task
TECO	4.1	—	15.3	—	—	—	20.3	2.8	42.5
WYLBUR	2.7	—	18.3	—	—	—	10.1	1.4	32.5
EMACS	2.0	2.5	4.6	—	—	—	7.8	6.9	23.8
NLS	2.5	3.0	4.3	2.0	—	1.0	4.9	1.3	19.0
BRAVOX	1.9	2.3	2.7	2.0	—	0.7	2.6	3.5	15.7
BRAVO	2.1	2.6	2.5	2.2	—	0.4	3.0	5.6	18.4
WANG	2.3	2.8	2.0	—	4.6	2.0	3.1	2.4	19.2
STAR	2.2	2.7	2.2	2.3	—	0.4	2.1	8.3	20.2
GYPSY	2.1	2.6	2.2	2.6	—	0.7	2.8	3.3	16.3

(b) Percentage of task time in each type of operation.

Editor	Operation Type							
	A	S	K	P	P_S	H	M	R
TECO	10%	—	34%	—	—	—	48%	7%
WYLBUR	8%	—	56%	—	—	—	31%	4%
EMACS	8%	10%	19%	—	—	—	33%	29%
NLS	13%	16%	22%	11%	—	5%	26%	7%
BRAVOX	12%	15%	17%	13%	—	5%	17%	22%
BRAVO	12%	14%	14%	12%	—	2%	16%	30%
WANG	12%	15%	11%	—	24%	10%	16%	13%
STAR	11%	13%	11%	11%	—	2%	10%	41%
GYPSY	13%	16%	14%	16%	—	5%	17%	20%

though the Time difference between these editors is not reliable with only four subjects per editor, this difference would be reliable if it had been found with ten subjects for each editor.

Importance. We want to emphasize the obvious fact that reliability is quite different from importance. Any observed difference between scores, however small, can be made reliable by running enough subjects. The real question is whether the observed difference is ***important***, which is a substantive, not a statistical, question. For example, small differences between editors on the Error dimension, even if they were reliable, may not be as important as the fact that the user population is highly variable; even large differences in the Time dimension would not be important in a situation where there were not many dedicated expert users.

In practical situations, small differences are usually not important, for they will be washed out by a host of interacting factors in the larger context. Thus, the fact that small observed differences are unreliable (except in extensive, expensive tests) is of little consequence. The utility of a relatively cheap test, such as the methodology proposed in this paper, is that it reveals potentially important (i.e., large) differences. Once a potentially important difference is identified, then it is a cost-benefit issue to determine how reliable the difference needs to be. But even if the difference is found to be reliable, it is not as important to be certain that there is some difference as to be certain that the difference is reliably large enough to matter.

One reason that the reliability issue arises is that only overall scores are being considered. Often, an informal visual inspection of the more detailed data comprising the overall scores can tell us more than a formal reliability analysis.

3.3 A Closer Look at the Data

The next several figures present breakdowns of the overall evaluation scores in Figure 6. Note that the editors in each of the figures are shown in different orders, corresponding to the order of scores on the different dimensions.

3.3.1. Time. Figure 7 is a scatter graph showing each individual expert user's error-free Time score. This graph shows the actual spread of user performance for each editor. The greater the overlap of the performance ranges of two editors, the less likely that the editors are reliably different. The individual points also allow us to identify outliers among the users. An outlier can penalize an editor's score compared to editors that were not unlucky enough to get an unusual user. For instance, the BRAVO outlier suggests that our mean is higher than it would be if the population of subjects had been larger and thus more evenly representative.

Also playing a part in the data is the mix of technical and nontechnical users run on each editor, since the tech-

nical users were on the average somewhat faster than the nontechnical users (this will be discussed in Sec. 4.1). We can adjust the overall editor scores to compensate for the different mix of technical and nontechnical users in each editor, but this adjustment does not change any score by more than 2 sec/task and turns out not to change the rank ordering of the editors.

CALCULATED TIME. The task times calculated with the Keystroke-Level Model [Figure 5(a)] are also shown in Figure 7. These calculated times correlate quite well with the empirical Time scores ($R = .90$). The calculated times are on average about 75 percent of the error-free Time scores (the worst case is 54 percent for BRAVOX, and the best case is 96 percent for STAR). The reader will note that there are two calculated times shown for TECO. The original prediction (shown in parentheses) predicted only about 49 percent of the actual error-free time. Because this calculation was so low and because we had time-stamped keystroke records of the users' actual behavior with TECO, we recalculated the task times using the actual methods that the subjects used (rather than trying to predict the methods, as we did for the original calculation). This second calculated time is 87 percent of the actual time. The discrepancy between the method predictions in the two calculations was due to the fact that the users were much more conservative, hence, less optimal, than predicted. The predicted methods used a minimum of searching, displaying, and verifying, while three of the four users were much more careful in their use of this nondisplay-based system. One user was much more daring, and the original calculation was about 70 percent of the actual time for that user—an outcome similar to the calculation results for the other editors.

The reasons for the rather consistent disparity between the Keystroke-Level Model calculations and the actual editing times have to do with the differences between the assumptions of the Keystroke-Level Model and the conditions of our experiments, as noted in Sec. 2.6. There are several differences: (1) The Keystroke-Level Mode assumes that the user's method for performing each task is known. However, we cannot always predict the methods, as we saw with TECO. We can usually predict the shorter, easier methods; but the longer, more complex methods are more difficult to predict. Since predicted methods are nearly optimal methods, when a user deviates from a

FIGURE 6. Overall Evaluation Scores for Nine Text Editors.

	Evaluation Dimensions			
Editor[a]	Time[b] $M \pm CV$[f] (sec/task)	Error[c] $M \pm CV$ (% Time)	Learning[d] $M \pm CV$ (min/task)	Functionality[e] (% tasks)
TECO	49 ± .17	15% ± .70	19.5 ± .29	39%
WYLBUR	42 ± .15	18% ± .85	8.2 ± .24	42%
EMACS	37 ± .15	6% ± 1.16	6.6 ± .22	49%
NLS	29 ± .15	22% ± .71	7.7 ± .26	77%
BRAVOX	29 ± .29	8% ± 1.03	5.4 ± .08	70%
BRAVO	26 ± .32	8% ± .75	7.3 ± .14	59%
WANG	26 ± .21	11% ± 1.11	6.2 ± .45	50%
STAR	21 ± .18	19% ± .51	6.2 ± .42	62%
GYPSY	19 ± .11	4% ± 2.00	4.3 ± .26	37%
$M(M)$ $M(CV)$[g]	31 .19	12% .98	7.9 .26	54%
$CV(M)$[g]	.31	.49	.53	.25

[a] The evaluations for TECO, WYLBUR, NLS, and WANG are from the first author's thesis [11]; the first author also evaluated STAR. The evaluations of the other editors were done in the second author's laboratory.

[b] The Time score is the average error-free expert performance time per benchmark task on the given editor. A difference between editors with mean values M_1 and M_2 is statistically reliable (95% confidence) if $|M_1 - M_2| > 0.33*(M_1 + M_2)/2$.

[c] The Errors score is the average time, as a percentage of the error-free performance time, that experts spend making and correcting errors on the given editor. A difference between editors with mean values M_1 and M_2 is statistically reliable (95% confidence) if $|M_1 - M_2| > 20\%$. Thus, no differences between editor means are reliable in this data.

[d] The Learning score is the average time for a novice to learn how to do a core editing task on the given editor. A difference between editors with mean values M_1 and M_2 is statistically reliable (95% confidence) if $|M_1 - M_2| > 0.45*(M_1 + M_2)/2$.

[e] The Functionality score is the percentage of the tasks in the task taxonomy (Figure 1) that can be accomplished with the given editor.

[f] The *Coefficient of Variation* (*CV*) = *Standard Deviation* / *Mean* is a normalized measure of variability. The *CV*s on the individual scores indicate the amount of between-user variability.

[g] The *M*(*CV*)s give the mean *between-user* variability on each evaluation dimension, and the *CV*(*M*)s give the mean *between-editor* variability on each dimension.

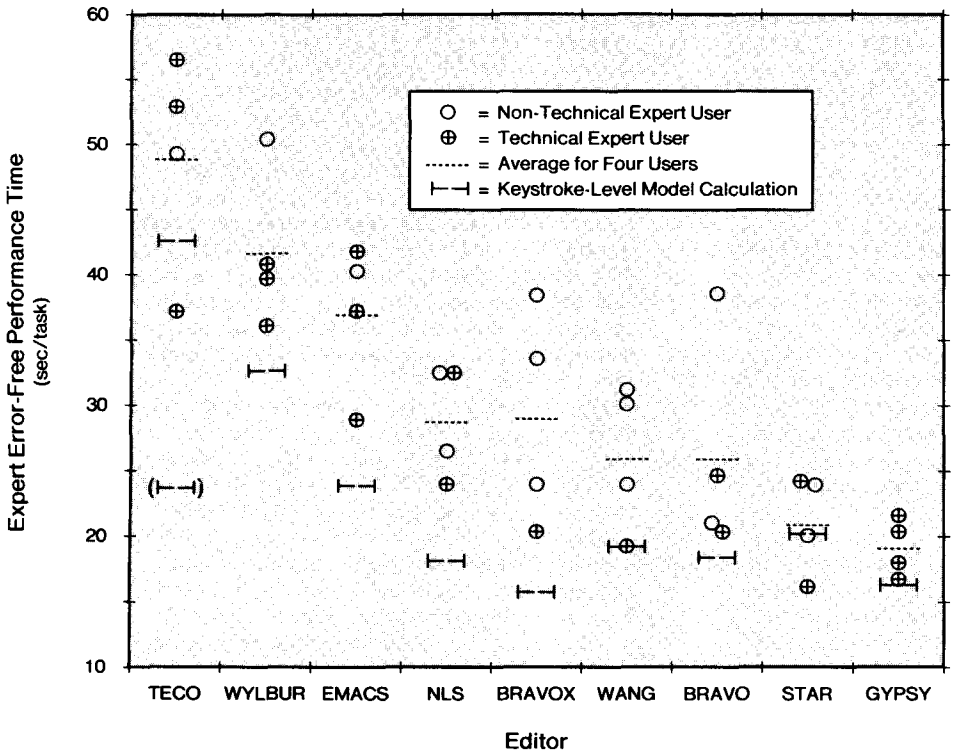

FIGURE 7. Error-Free Time Scores for Individual Expert Users. The editors are ordered by descending Time score.

predicted method, it is usually in the direction of using a slower method. (2) Some of the users may have had to engage in problem-solving to perform some of the more complex tasks in some editors (e.g., to transpose phrases with TECO) and their behavior would not be the simple method-execution behavior assumed by the model. (3) The error time for small errors is included in the experimental error-free time, but is not considered in the calculated time. (4) The experimental time includes all the time ***between*** tasks. Some of this time is not considered in the model, such as page turning time, pauses for rest, etc. But even without such differences, it should be remembered that the Keystroke-Level Model is an approximate model, and we should not expect its calculations to be perfect.

The data for individual users show that, for most editors, one user comes very close to the level of performance represented by the Keystroke-Level Model calculation. Since the calculations were based on predictions of optimal methods, this suggests that only a minority of users are likely to approach optimal performance.[10]

[10] The one exception to this observation is in STAR, which had one user who performed much better than the Keystroke-Level Model calculation. We believe that that is because the user constantly overlapped his actions with STAR's long system response times; he often did not wait for the machine to catch up with him between tasks, but typed ahead whenever possible.

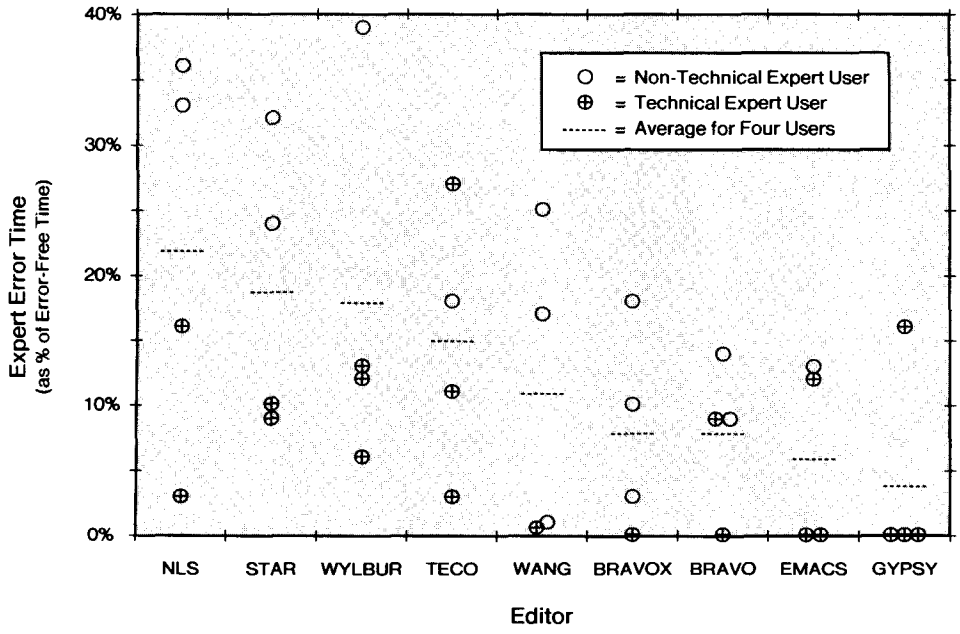

FIGURE 8. Error Time Scores for Individual Expert Users. The editors are ordered by descending Error score.

3.3.2. Error. Figure 8 is a scatter plot of the individual expert users' Error scores. This data shows a factor of 5 difference between the best and the worst editors; even so, these differences are swamped by the large ranges of error within editors. The relative variabilities are summarized in Figure 6: the between-editor *CV* is .49, whereas the between-user *CV* averages .98. Thus, no conclusions can be drawn about the differences between editors in error cost.

It might be noted that the individual users who have large Error scores do not have them because they were unfortunate enough to be struck by rare, disastrous errors; rather, these users merited their Error scores by committing several errors throughout the experiment. Among the seven users whose Error scores were greater than 20 percent, the error time came from an average of 7.4 individual errors: 3.1 during the first pass over the benchmark and 4.3 incomplete tasks that had to be fixed up on the second pass. The errors during both passes took an average of over 70 seconds each.

3.3.3 Learning. The overall Learning scores are broken down in two ways: by time and by individual learners. Figure 9 gives learning curves over time for all of the editors, each curve being the average of four learners. Each learning curve is drawn in a stylized fashion as a series of five steps, one step for each cycle in the learning session. The instruction part of a cycle is represented by the sloped part of the step, and the quiz part of the cycle is represented by the flat part of the step (as if no learning occurs during the quiz). These curves can be seen to be fairly straight overall, indicating that it is reasonable to summarize them using their overall slopes, which is just what the Learning scores are.

The reader will note that there are two learning curves for TECO. The learning test was replicated for TECO with a second instructor, who ran the test completely independently. The second instructor, using only the materials in [11], taught a slightly different set of TECO commands than the first instructor and of course taught a different set of four subjects. The results of this second evaluation test (marked JF) can be seen to be quite close to the first (marked TR).

Figure 10 is a scatter graph of the individual novices' Learning scores. This graph, as well as Figure 9, shows large differences in the learnability of the different editors. TECO is clearly different from all the others, taking over twice as long to learn as the next editor (WYLBUR). The rest of the editors lie in a tight group with considerable overlap between adjacent editors. But this group still covers another factor of two in learning time, so GYPSY is four times as fast to learn as TECO. The large amounts of overlap in the range of learners within editors indicate that the differences between adjacent editors are mostly not reliable. The difference in Learning scores between TECO and WYLBUR is reliable, as are the differences be-

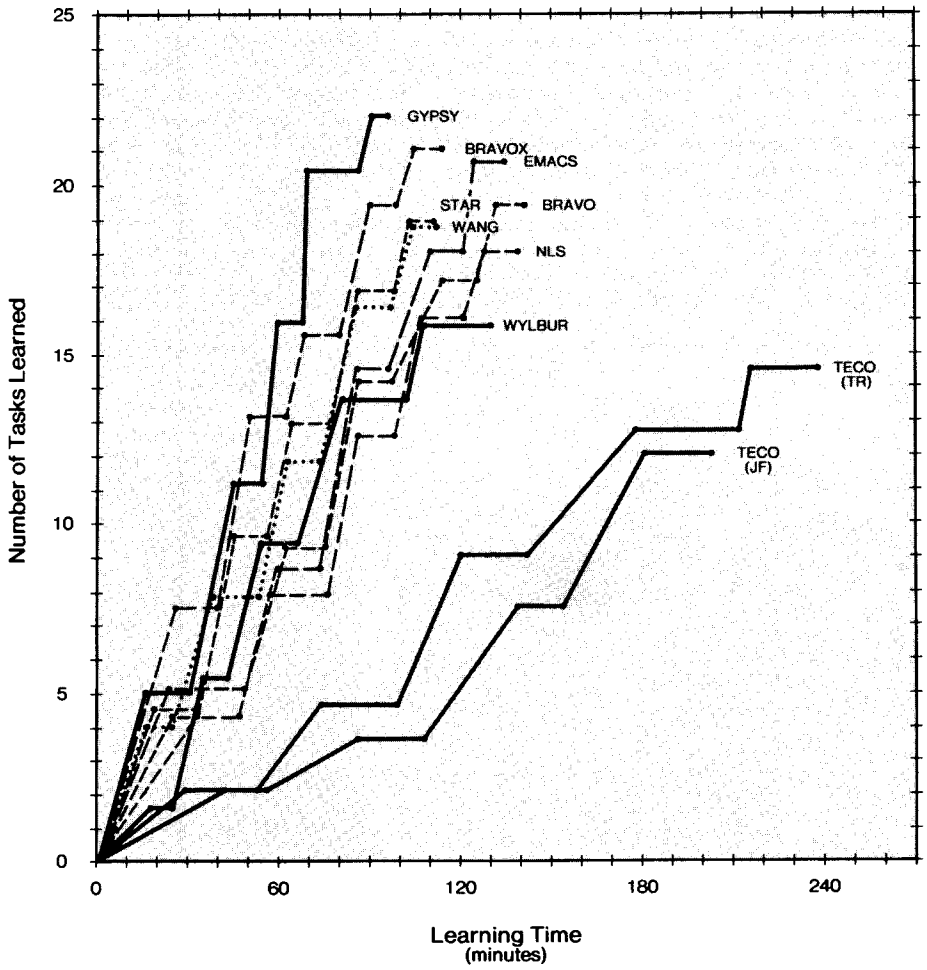

FIGURE 9. Average Learning Curves over all Learners on each Editor. The two TECO curves were produced by different instructors.

tween GYPSY and each of TECO, WYLBUR, NLS, and BRAVO.

Figure 10 allows us to identify outlier learners, as we did with the Time scores. One such outlier is a STAR learner, which suggests that the mean Learning score for STAR might be slightly lower from a more representative subject sample. In addition, there was one subject who was completely unable to learn TECO at all (that subject's partial data is not included in any of our data or graphs). The fact that the only learning failure of the whole set of learning experiments occurred with TECO reinforces the notion that TECO is more difficult to learn than the rest.

INSTRUCTOR EFFECTS. The instructor plays a strong role in the learning experiments—s/he decides what subset of commands to teach, and s/he tries to maximize the learning rate by keeping the subject from getting bogged down in nonproductive efforts. Thus, the instructor could have a potentially strong effect on the learning results. To show instructor effects, the specific instructors are noted in Figure 10. Since the scores for the different editors overlap so much, it seems that no instructor is consistently faster or slower than the others. This can be seen most clearly in the cases where the learning tests have been replicated. In the TECO case (mentioned above), the second instructor obtained a mean Learning score within 12 percent of the score obtained by the first instructor. In the second case, the EMACS learning tests were replicated in a different laboratory, obtaining a virtually identical overall Learning score [13].

The differences in teaching style of the different instructors can, on the other hand, be seen in the between-subject variations. The two TECO data sets show this difference most clearly—the second instructor has very much less between-subject variation. This can also be seen in the between-subject *CV*s in the editor evaluations run by TR and BS, the two instructors who ran most of the tests. TR's *CV*s range from .24 to .45, while BS's *CV*s range from .08 to .26. The instructors seem to be exerting different amounts of control over the learners. However, this does not seem to affect the mean Learning scores.

3.3.4. Functionality. Figure 11 gives a breakdown of the Functionality scores by the different categories in the task taxonomy. These functionality results show that most of the editors can perform about half of the tasks in the task taxonomy. Each system has its areas of strength and weakness. To show this, the scores are broken down into subscores in Figure 11. For instance, EMACS is excellent in programming capability, while NLS and BRAVOX are especially good in formatting and layout tasks. Because the number of tasks in the taxonomy was weighted more toward text layout than programming, the document-oriented editors generally scored somewhat better overall than EMACS. But NLS, which tries to cover all needs, is clearly superior in overall functionality.

We can question the reliability of these Functionality scores, as well as the other scores generated by this methodology. An analyst's rating of the functionality of an editor is partly a matter of judgment, as was noted in Sec. 2.5, and partly a matter of detailed knowledge of the edi-

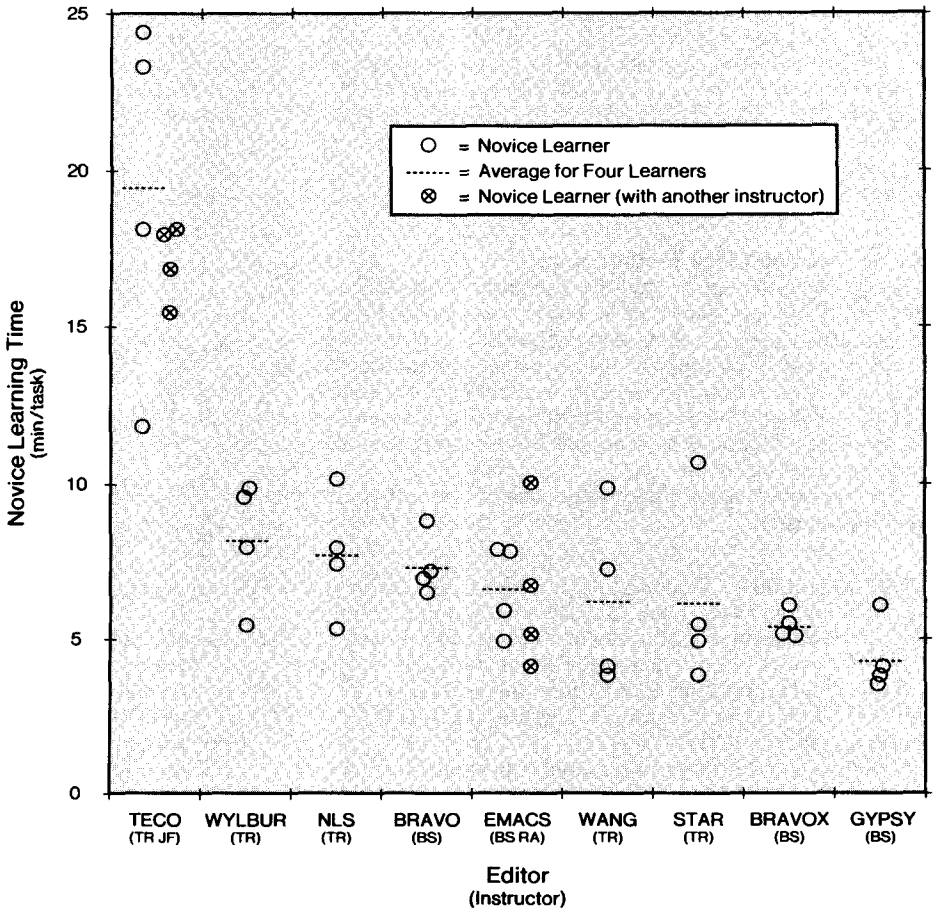

FIGURE 10. Learning Scores for Individual Novice Learners. The editors are ordered by descending Learning score. The instructors are noted below each editor.

FIGURE 11. Functionality Subscores for the Nine Text Editors.

Task Category[a] (No. of Tasks)[b]	Editor[c] NLS	BRAVOX	STAR	BRAVO	WANG	EMACS	WYLBUR	TECO	GYPSY	All Editors[d] M±CV
TOTAL (212)[b]	77%	70%	62%	59%	50%	49%	42%	39%	37%	54%±.25
Modification										
Content (66)	94%	89%	93%	90%	87%	74%	63%	88%	80%	84%±.13
Text Layout (19)	89%	71%	66%	71%	37%	37%	26%	3%	26%	47%±.56
Page Layout (25)	74%	62%	56%	40%	34%	2%	6%	4%	4%	31%±.85
Characters (21)	43%	76%	57%	62%	38%	14%	21%	0%	17%	36%±.66
Other (16)	53%	59%	50%	22%	34%	0%	16%	3%	0%	26%±.84
Addressing (22)	68%	36%	30%	30%	16%	61%	34%	25%	18%	35%±.48
Control (23)	56%	37%	24%	20%	24%	89%	61%	48%	9%	41%±.58
Display (8)	94%	94%	63%	69%	19%	81%	62%	38%	50%	63%±.42
Misc. (12)	100%	88%	100%	71%	71%	46%	71%	25%	42%	68%±.38

[a] The Task Categories are described in the task taxonomy shown in Figure 1.

[b] The number in parentheses after the task category name gives the total number of tasks in that task category. The Functionality scores and sub-scores are given as a percentage of the total number of tasks in each task category. The scores in the TOTAL row are the same as in Figure 6.

[c] The editors are ordered in descending order of their overall Functionality scores.

[d] The numbers in the All Editors column tell how well the task categories are handled by the whole collection of editors and the amount of between-editor variability there is.

tor (e.g., knowing about limitations that may not be apparent from the documentation). To quantify the variation between analysts, three different analysts were asked to independently rate WYLBUR. The overall Functionality scores for the three analysts were 42 percent, 45 percent, and 39 percent. Scores within task categories differed more, but the differences between the analysts tended to be averaged out over the total set of tasks. Thus, as a rule of thumb, we can consider the overall Functionality scores to be accurate to around 10 percent.

3.4 Assessment of the Methodology

The above results show that diverse editors can indeed be evaluated and compared. As a whole, the evaluation methodology seems to successfully provide an objective, multidimensional picture of text editors. This methodology is also quite practical. For an experienced evaluator, about one week of time is required to evaluate a new editor. Thus, it should be practical for a system designer or a potential buyer.

Several other issues surrounding the methodology deserve discussion.

3.4.1. Reliability. The main drawback in the use of this methodology is that the small number of subjects used for each of the tests makes the results very coarse. In addition, the results point out that the Error dimension needs a more reliable measure to differentiate editors, which will have to take into account the effect of large differences among the users.

Another way to increase reliability, besides increasing the number of subjects, is to decrease the between-user variability by homogenizing the subject sample. For instance, potential subjects could take a pretest, and only people who scored within a certain range could be used. This, however, specializes the results so that they only represent a small segment of the user population, decreasing the generality of the methodology.[11]

Given that the methodology accepts a wide range of subjects, we can check whether the methodology is being applied to a restricted sample. If the between-user variance is ever substantially less than in the data here, the reason may be that the evaluator has picked a restricted sample of subjects. This is a useful caution for designers who are testing their own systems and who especially have to guard against bias. For example, in the data presented here, we note that the Time data for GYPSY does in fact have a lower than normal *CV*, which in this case is largely explained by the fact that only technical subjects were used.

3.4.2. Coverage. Although the methodology covers several basic aspects of editor usage, there are still aspects not covered. When this methodology was being developed [11], a variety of easy-to-obtain measures of other aspects were explored. Some examples are: (1) The error-proneness of an editor was measured by putting external stress on expert users while they performed editing tasks. (2) The possibility of disastrous errors in an editor was measured by a procedure for analyzing the editor's command language. (3) The display capabilities of an editor were measured by users performing proofreading tasks. (4) The learning and use of advanced features was addressed by using a questionnaire to measure experts' knowledge of how to perform complex editing tasks. Unfortunately, all of these attempts turned out to be too crude to be reliable and too unproductive in differentiating systems. The tests presented in this paper are the only ones we know currently that work well enough to be included in a methodology.

[11] Another way to increase reliability is to use all the subjects, but to use the pretest scores to normalize the overall results. This would require a model of the relation between pretest scores and performances results.

3.4.3. Representativeness.

TIME. A general criticism of benchmark testing is that the items in the benchmark are not appropriate or appropriately weighted for any particular application. Specifically, the benchmark used in the present methodology has been criticized for not representing the true mix of tasks in real text-editing situations [17]. This is true, as we noted in Sec. 2.2. However, we are skeptical that there is a single benchmark set representing the majority of text-editing situations. This is an empirical issue, and we know of no data currently that settles it. But there remains the issue of how to use the results of the present methodology if one is interested in a particular situation that has a different mix of tasks from the benchmark.

We propose an analytic procedure for adjusting the Time score from the benchmark test to correspond to a new situation, which is characterized as a new set of tasks (weighted by the frequency of the individual tasks). This adjustment procedure is based on the assumption that there is a constant ratio between the experimentally measured Time score and the time calculated with the Keystroke-Level Model. This can be expressed in a formula:

$$T/C = T'/C'$$

where T is the Time score on the benchmark, C is the calculated time on the same benchmark (as in Figure 5), C' is the calculated time for the new mix of tasks, and T' is the Time score we would expect from an experimental test on the new mix of tasks. T and C are given by the present methodology. T' is the desired result. It can be estimated by calculating C', which is done by using the Keystroke-Level Model on the new (weighted) set of tasks. One must be cautious about the assumption behind this adjustment procedure, especially if the new task set contains many complex editing tasks, for the assumptions behind the Keystroke-Level Model (see [3]) might be violated (such as was our experience with the first TECO calculation).

LEARNING. The particular set of tasks chosen for the learning experiments undoubtedly affects the results obtained here, but it is likely to be less influential than which teaching paradigm is used. For example, we would expect the results of a self-teaching paradigm to be mostly determined by the quality of documentation. We do not in general know how teaching paradigms differ, but there is one preliminary result in a recent study by Robertson and Akscyn [13] comparing different teaching paradigms. They applied the present learning methodology to the ZOG frame editor, using an instructor and using two self-teaching paradigms by substituting online and offline documentation for the instructor. They found that the instructor produced about 13 percent faster learning than the self-teaching documentaton; and they found that the offline documentation was about 6 percent faster than the online documentation. The reason for the small difference caused by mode of documentation was that all the learners used the documentation in the same way in both cases—by reading through it at the beginning of the session. The lesson here is that real learners do not necessarily follow the paradigms laid out for them by the system documenters.

FUNCTIONALITY. Finally, the issue of representativeness also applies to the checklist of tasks for testing functionality: the tasks in the checklist do not represent the needs of any particular situation. The degree of elaboration of the tasks in the task taxonomy was influenced by the capabilities of the editors existing or being envisioned at the time the taxonomy was being created. Thus, there are eight tasks relating to the layout of paragraphs but only one about the ability to typeset mathematical formulas properly. An editor that performs both functions equally well gets far more credit for one than the other. This problem is best addressed by using the functionality subscores; for a given application more weight can be given to the areas relevant to the application.

3.4.4. Applicability.

EXTRAPOLATION TO A LARGER CONTEXT. All of the data we have gathered have been from people performing a small number of preset tasks in a laboratory environment. What relationship do these results have to productivity in an office where the tasks may be different (e.g., proofreading and editing one's own work) and the environmental conditions may be different (e.g., a receptionist with constant small interruptions from people walking by)? A 20 percent improvement in the Time score for this methodology would not necessarily translate into a 20 percent improvement in overall office productivity. This is because an improvement in editing speed may not be accompanied by a proportional improvement in the speed of other activities that the user is doing along with editing, such as thinking about the proper wording of the text, typing in large amounts of new text, or proofreading for errors. Another possible factor is that the intense concentration on the editing task allowed by laboratory conditions, but often not allowed by real situations, may differentially affect the performance of different editors. Such problems beset all laboratory work, and the questions raised can only be answered when laboratory studies are supplemented by on-site studies to determine the relationship between the two.

USE BY EDITOR DESIGNERS. The full methodology requires an implemented text editor that has been running long enough to have at least a few expert users, which suggests that the methodology is not useful for a designer of a new editor. However, the designer can use parts of the methodology to get an early indication of how well the proposed editor compares with existing editors and where the strengths and weaknesses of the new editor lie. Two of the evaluation measures, Time and Functionality, can be obtained analytically, when the design is still on paper. Learning can be measured experimentally on a prototype (that need only be complete enough to cover the core tasks). The Error measure is the only one that cannot be obtained easily; this should pose no problem, since editors cannot be differentiated on this dimension anyhow.

On the Time dimension, the Keystroke-Level Model can be used to produce a calculated task time, along with a decomposition of the time into the times for the different operations. These times can be compared to the calculated task times for other editors in Figure 5 to see whether the times are in line with similar editors and to reveal possible bottlenecks on some operations. (The calculated task time can also be adjusted, by multiplying by 1.3, to compensate for the model's tendency to underpredict the experimental Time scores. The adjusted time can then be compared to the Time scores in Figure 6). In this analysis, the only parameters which must be estimated

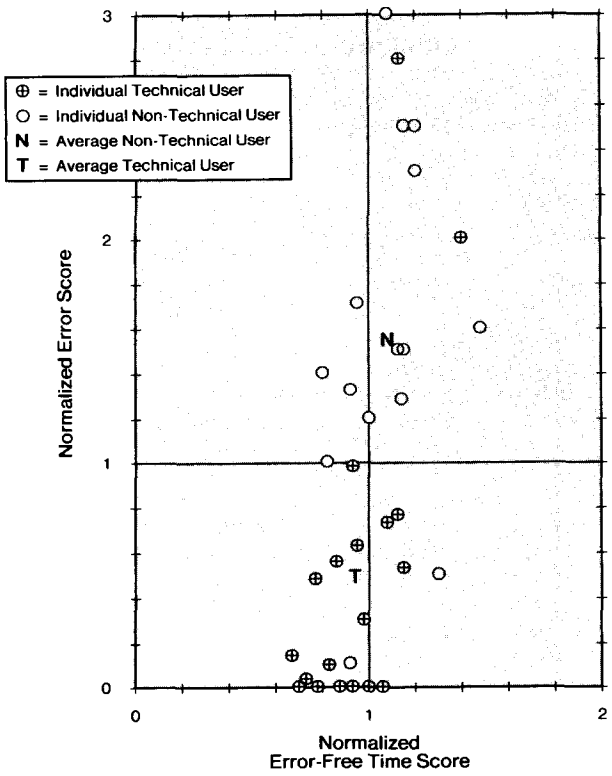

FIGURE 12. Normalized Time and Error Scores for all Expert Users. A user's score is normalized by dividing it by the average score in the editor.

are the system response times. If these are not available, this analysis can be turned around to provide the designer with a specification for acceptable limits for the response times (by showing how different response times make the proposed editor compare to other editors). Finally, if a prototype system is available, experimental benchmark tests can be run using the designers and implementors themselves as subjects. These data would be useful to provide a check on the calculated times and the predicted methods that the times are based on.

4. BEHAVIORAL RESULTS

The database of results from the experimental studies gives us information not only about the specific editors, but about user behavior in general, such as the gross levels of user performance in text editing. The data show that the core editing tasks require about 20–45 seconds per task for most expert users on most systems, and it shows that a period of about two hours of one-on-one training is enough to teach novice users about 20 core tasks in most editors. These results should be of interest to researchers in office productivity, for example, to measure the cost-effectiveness of word processing. More detailed results are interesting in two principal ways: for the light they shed on (1) the individual differences in performance between users and (2) the factors influencing novice learning.

4.1 Individual User Differences

4.1.1. Magnitude of Individual User Differences. The greatest individual differences by far are found in Error time scores (ranging from 0 to 39 percent), which reflects a wide variation among expert users in how careful they are in avoiding errors and in performing tasks completely. There is much less variation among experts in speed of editing—about a factor of 1.5 to 2 between the fastest and slowest users' Time scores within each editor. This range is much smaller than the factor of 3.5 reported in [4]. However, [4] tested a more diverse sample of users, including casual users as well as dedicated expert users.

A somewhat surprising result is that the variation among novice learners is not much greater than among expert users. Learners exhibit about the same range of variation (up to a factor of 2.5 between the fastest and slowest learners within an editor) and CV (.19 for experts and .26 for novice learners). This is partly due, no doubt, to the fact that the learning tests are designed to minimize variation due to idiosyncratic learners (e.g., the command summary sheet and the always present instructor). A self-teaching paradigm is likely to yield much more variation among learners.

4.1.2. Time vs. Errors. It is common wisdom that there is a speed-accuracy tradeoff: that when people work faster, they make more errors. Our data can be used to investigate whether the users who spend more time in error do so because they are working faster, that is, whether users with higher Error time scores have lower error-free Time scores. We cannot directly compare scores of users on different editors, however, unless we normalize over editors. A user's score on an editor can be normalized by dividing it by the overall (mean) score for the editor. That is, a normalized score of 1.0 indicates an average user, and a score of .5 indicates a user twice as good as the average. Figure 12 plots the normalized Time vs. Error scores for all the expert users. What is immediately obvious from this plot is the much larger variation on the Error dimension than on the Time dimension. However, we do not see the tradeoff between Time and Error scores that a speed-accuracy tradeoff would suggest, but rather a modest positive correlation between them ($R = .58$). Some users tend to be better than others on both dimensions.

4.1.3. Technical vs. Nontechnical Expert Users. The individual users plotted in Figure 12 are marked as being technical or nontechnical. The technical users are clearly the better users on both Time and Error (clustering in the lower left quadrant). Also plotted in the figure is the average technical user and the average nontechnical user. These two fictitious average users account for the major features of the plot. The average nontechnical user is 15 percent slower than the average technical user (.94 vs. 1.08) and spends a factor of 3 more time in error (.50 vs. 1.56).[12] The factor 1.15 difference between technical and nontechnical users on the Time dimension is comparable to the factor of 1.3 reported in [4].

The underlying reason for the difference between technical and nontechnical users is not known. It is not due to physical skill factors, such as typing proficiency, for which nontechnical users are superior.[13] It could just be

[12] This data allows us to calculate an adjustment for the effects of using different proportions of technical and nontechnical subjects in different editors. As mentioned in Sec. 3.3.1, such an adjustment does not change the rank ordering of the editors of the Time dimension. A similar adjustment on the Error dimension also makes little difference in the results: the range of Error scores becomes a factor of 4 instead of a factor of 5, and differences between editors are still not statistically reliable.

[13] The nontechnical users were 1.4 times faster than the technical users. Given that an average of about 22 percent of the time is spent in typing [Figure 5(b)], this would give the nontechnical users about a 7 percent advantage over the technical users.

due to a difference in general intelligence or education, rather than anything due to technical experience per se. (The programmers we used as technical subjects have been preselected to be very bright and highly educated, whereas the secretarial and support personnel we used as nontechnical subjects have undergone less of such preselection.) Other possible factors, suggested by a recent study [5], are that technical users might have more spatial ability or be younger than nontechnical users. These two factors have been shown to affect editor learning rates, and they are also likely to apply to expert performance.

4.2. Novice Learning

Learning behavior is less well understood than expert performance. The Keystroke-Level Model [3] (along with its theoretical underpinnings [4]) provides a usefully accurate account of the time performance of expert users. However, we have no similar account of why some editors are easier for novices to learn than others. Our learning data provide the opportunity to test some ideas about the main factors affecting learnability.

*4.2.1. **Factors Affecting Editor Learnability.*** How does the structure of an editor affect its learnability? Perhaps the most obvious hypothesis to consider is that the command languages of some editors are more complex. One measure of command language complexity is the number of distinct commands in an editor. According to this hypothesis, the editors with fewer commands should be faster to learn. (This might be called the "weigh-the-manual" theory of learnability, since most reference manuals consist of an enumeration of the different commands.) In this methodology, since only commands necessary to do core editing tasks are taught, we restrict our measure to the number of these "core commands." Figure 13 shows that this measure correlates poorly ($R = .37$) with the Learning scores.[14]

FIGURE 13. Correlations of Learning Scores with Various Measures.

Measure	Correlation (R)	
	All Nine Editors	All Editors except TECO[a]
Number of Core Commands in Editor	.37	.19
Number of Physical Operations per Task	.68	.58
Number of Method Chunks (M's + A's) per Task	.93	.65
Expert Time Score	.79	.67

[a] Since the Learning score for TECO is an extreme value, it has a large influence on the correlations. Hence, it is useful to present a separate set of correlations with the influence of the TECO score removed.

The crucial point missed by this hypothesis is that commands are not useful in isolation, rather they are used in the context of methods or procedures to accomplish editing tasks. Thus, the second hypothesis to consider as a predictor of learnability is that learning is related to the ***procedural complexity*** of a command language. This is quite different from command language complexity. For example, a "simple" command language with only three commands might require lengthy and intricate procedures to accomplish editing tasks, whereas an editor with a large variety of commands might only require a couple of those commands to do any one task. The procedural complexity hypothesis says that a user must learn not just what each command does, but how each command is used in various ways in different methods. This leads us to consider the number of distinct uses of commands, which is related to the length of the methods (rather than the length of the list of commands).

One way to approximate the procedural complexity of an editor is to compute the average number of steps in the methods for accomplishing a representative set of tasks, such as the benchmark used in the Time and Error dimensions of our methodology.[15] The physical operations in the Keystroke-Level Model encodings of methods (see Figure 4) provide a simple, unambiguous set of steps to count. Figure 13 shows that the average number of physical operations per task correlates substantially better with the Learning scores ($R = .68$) than do the commands, although the correlation is still modest.

The length (in physical operations) of a method, although it may correlate with procedural complexity, can be a misleading indicator. For example, a method requiring the user to type D E L E T E RETURN is not seven times more complex than a method requiring only D to be typed. Thus, we see that procedural complexity has more to do with the ***mental*** "chunking" of physical steps into coherent fragments than the physical steps themselves. To operationalize this notion, let us return to the Keystroke-Level Model encoding of methods. This model has two kinds of mental operations, **A**'s and **M**'s. When a large editing task is broken into subtasks, the subtasks are each preceded by an **A** operation, representing the user's having to acquire a mental representation of the subtask. Within a subtask, the sequence of physical operations is punctuated with **M** operations, which represents small mental preparations for the upcoming physical operations (rules for placing **M** operations are given in [3]). The **A** and **M** operations have the effect of breaking the sequence of physical operations into procedural chunks. For example, consider the method encodings of the example task in Figure 4. The method for WYLBUR is:

A M 12K **M** 3K R **M** 13K **M** 20K **M** 8K **M** K

Here the physical operations are divided into seven chunks by the **A**'s and **M**'s. The methods for the same task in EMACS and STAR contain only three and two chunks, respectively.

A S **M** 4K R 2K **M** 15K
A S H P 2K **M** K H 14K

The number of chunks in a method, which can be estimated by simply counting the **A**'s and **M**'s, should be a better indicator of the procedural complexity of the method than the physical operations we counted before. In fact, the mental chunking measure correlates better with the Learning scores ($R = .93$) than do the physical operations, as Figure 13 shows. It is the best correlate we have of Learning time.

This notion of procedural complexity as determined by mentally defined chunks is an instance of the "zeroth-order theory of learning" [4]: that learning time is proportional to the number of chunks of information that must

[14] One problem with this measure is deciding what a command is (e.g., is a preselection a command itself or just an argument to a command that follows it?). This issue can be sidestepped somewhat by counting parts of commands, such as commands names, arguments, terminators, etc. However, this "finer" measure does no better than just counting "whole" commands (see [11]).

[15] It may seem paradoxical that we are using the expert benchmark test to measure learnability by novices. But note that we are only using the benchmark test as a convenient sample of tasks to get at the procedural complexity *required by the core functions of the editor.* Since the novices are trying to acquire this same expertise, it represents the *target competence* they are trying to achieve.

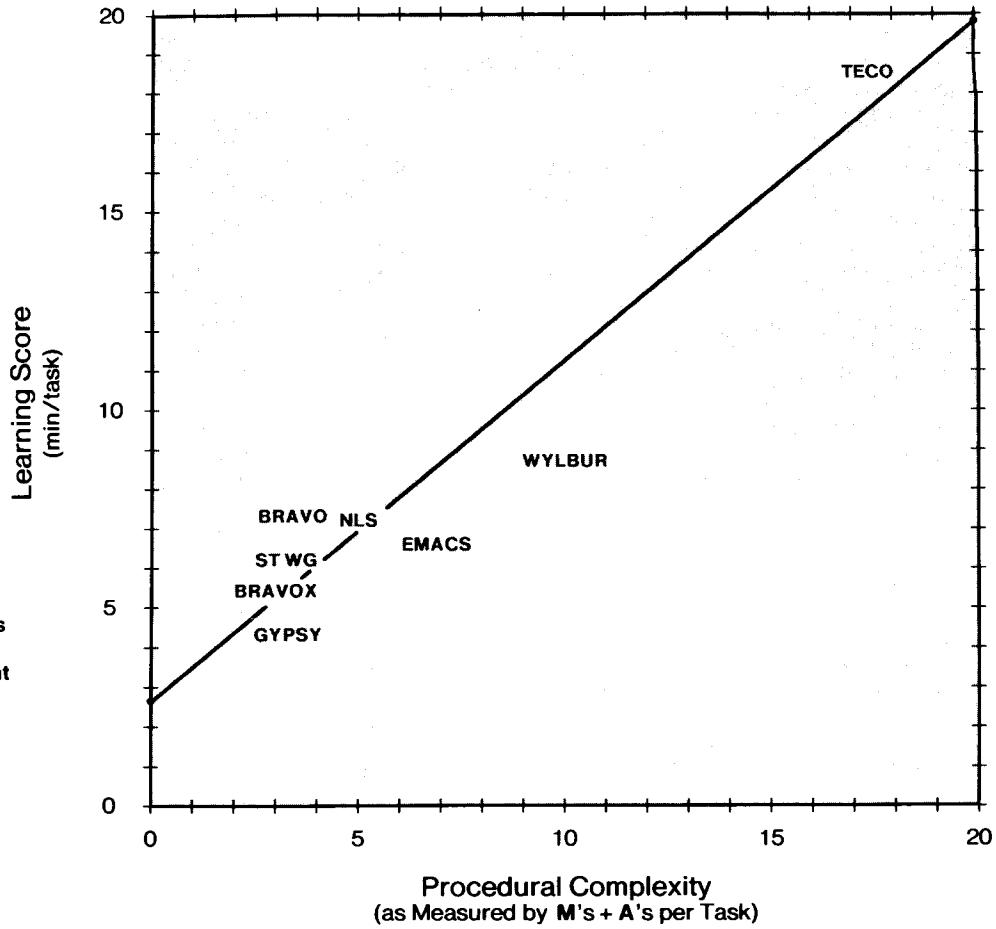

FIGURE 14. Plot of Learning Scores vs. the Mental-Chunks Measure of Procedural Complexity. (Note that STAR is abbreviated as ST and WANG as WG.)

be learned. To make this theory operational, we must be able to specify what the chunks are. In this case, the chunks are the procedural fragments bounded by mental operations.

Figure 14 shows a plot of the mental-chunking measure of procedural complexity against the Learning scores. This plot shows how raw correlations must be interpreted with caution, for we see that the learning score for TECO, which lies far out from the others, has great leverage on the correlation (which is why we also give the correlations excluding TECO in Figure 13). What we see in Figure 14 is that procedural complexity accounts for the difference between the fastest and slowest editors, but that it tells us little about the observed differences among the set of fastest editors. Procedural complexity is not the only factor affecting learnability; in fact, it seems to be dominated by other factors among fast editors.[16] However, procedural complexity may be the most dominant factor in learning overall, statistically accounting for about half the variance between editors.

4.2.2. Learning vs. Time. The conventional wisdom among designers is that there is a tradeoff between systems that are easy to learn by novices and systems that are efficient to use by experts. However, if we correlate the Learning scores with the Time scores, we see exactly the opposite. The data from our study shows a high positive correlation ($R = .79$, Figure 13) between the Time and Learning scores.[17] The concept of procedural complexity introduced in the last section explains this correlation. It says that the same factor—procedural complexity—underlies both expert performance (longer methods take longer to execute) and novice learning (longer methods imply that there are more chunks to learn).[18]

5. CONCLUSION

A standardized four-dimensional methodology for evaluating text editors has been presented and applied to nine different editors. The methodology seems to be an effective tool for the empirical evaluation of text editors along the dimensions of Time, Error, Learning, and Functionality. Of course, the methodology has limitations—having to do with reliability, coverage, representativeness, and applicability—which is the price of keeping the methodology simple to use. It is obvious that the methodology could be improved by both refinement and extension.

[16] A candidate for one of the other factors is what we might call "conceptual unfamiliarity," which taps how well novice users understand, *a priori*, the conceptual constructs involved in an editor. This notion is currently being explored by the second author and Sally Douglas in *Learning to Text Edit: Semantics in Procedural Skill Acquisition*. Ph.d dissertation, Stanford University, March '83.

[17] This is the only substantial correlation between scores on the methodology's dimensions. Correlations between the other dimensions range between .24 and .36. All of these correlations are positive, in the sense that editors tend to improve in the two dimensions together, with the exception that there is a tradeoff between Error and Functionality.

[18] We can use this result to conjecture that the main reason for the superiority of display-based systems, on both the Time and Learning dimensions, over nondisplay systems is not the display itself, but rather that the display-based systems permit much less complex procedures.

However, even in its present form, it provides for the generation of a valuable user-editor performance database of objective measures. We would urge others who need to do evaluations of editors to use this methodology. Its main advantage is that the numbers produced can be put in the context of the database of already evaluated editors (without such a context, numbers are difficult to interpret). At the same time, the additional evaluations (either replications of existing evaluations or evaluations of new editors) would be contributing to extending the database, allowing our knowledge of editor performance to systematically accumulate.

We have also shown how the database of results can help us understand user performance, by making clear the magnitude of individual differences of both experts and novices, and by providing a testing ground for understanding the factors affecting learning. Although we presently favor a theory of learning based on the notion of procedural complexity, a larger database will show whether this theory holds up. Finally, we have shown that Keystroke-Level Model calculations of editor performance, which also belong in the database, are useful analyses against which to compare and interpret the experimental results.

Acknowledgements. We thank Betsey Summers for organizing and running many of these evaluation studies and for helping us analyze the data. We thank Allen Newell and Stu Card for many helpful discussions and Allen Newell for commenting on drafts of this paper.

This paper is based in part on the first author's thesis research (reported in [11]) which was done under the supervision of the second author, and on continuing research by the second author. A short, preliminary version of this paper was published as [12]. The authors are listed in reverse-alphabetic order.

REFERENCES

1. Boies, S. J. User behavior in an interactive computer system. *IBM Systems Journal 13* (1974) 1-18.
2. Card, S. K., English, W. K., and Burr, B. J. Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys for text selection on a CRT. *Ergonomics 21* (1978) 601-613.
3. Card, S. K., Moran, T. P., and Newell, A. The Keystroke-Level Model for user performance time with interactive systems. *Comm. ACM 23, 7* (July 1980) 396-410.
4. Card, S. K., Moran, T. P., and Newell. A. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.
5. Egan, D. E., Bowers, C., and Gomez, L. M. Learner characteristics that predict success in using a text-editor tutorial. *Proc. Conference on Human Factors in Computer Systems*, Gaithersburg, MD, (March 1982), 337-340.
6. Embley, D. W., and Nagy, G. Behavioral aspects of text editors. *Computing Surveys 13*, 1 (March 1981) 33-70.
7. Good, M. An ease of use evaluation of an integrated document processing system. *Proc. Conference on Human Factors in Computer Systems*, Gaithersburg, MD, (March 1982), 142-147.
8. Meyrowitz, N., and van Dam, A. Interactive editing systems. *Computing Surveys 14*, 3 (Sept. 1982) 321-415.
9. Moran, T. P. The Command Language Grammar: A representation for the user interface of interactive computer systems. *Int. Journal of Man-Machine Studies 15*, 1 (July 1981) 3-50.
10. Riddle, E. A. *Comparative Study of Various Text Editors and Formatting Systems.* Report AD-A029 050, Air Force Data Services Center, The Pentagon, Washington, D.C., (Aug. 1976).
11. Roberts, T. L. *Evaluation of Computer Text Editors.* Ph.D. dissertation, Department of Computer Science, Stanford University, Stanford, Calif., (1980). Available as Report AAD 80-11699 from University Microfilms, Ann Arbor, Mich.
12. Roberts, T. L., and Moran, T. P. Evaluation of text editors. *Proc. Conference on Human Factors in Computer Systems*, Gaithersburg, MD, (March 1982), 136-141.
13. Robertson, C. K., and Akscyn, R. Experimental evaluation of tools for teaching the ZOG frame editor. Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, (1982).
14. *The Seybold Report on Office Systems* (through 1981 called *The Seybold Report on Word Processing*). Media, PA.
15. *The Seybold Report on Word Processing.* 4, 4, (April 1981). Issue on Personal Computers: Word Processing Packages.
16. Smith, D. C., Irby, C., Kimball, R., Verplank, W., and Harslem, E. Designing the Star user interface. *Byte 7*, 4 (April 1982) 242-282.
17. Whiteside, J., Archer, N., Wixon, D., and Good, M. How do people really use text editors? *Proc. SIGOA Conference on Office Information Systems*, Philadelphia, (1982) 29-40.

References (for Editor Documentation)

18. Augmentation Research Center. *NLS-8 Command Summary.* Stanford Research Institute, Menlo Park, Calif., (May 1975).
19. Augmentation Research Center. *NLS-8 Glossary.* Stanford Research Institute. Menlo Park, Calif., (July 1975).
20. Bolt, Beranek, and Newman, Inc. *TENEX Text Editor and Corrector* (Manual DEC10-NGZEB-D). Cambridge, Mass., (1973). (Documents TECO.)
21. Garcia, K. *Xerox Document System Reference Manual.* Xerox Office Products Division, Palo Alto, Calif, (1980). (Documents BRAVOX.)
22. Lampson, B. Bravo manual. *Alto User's Handbook.* Xerox Palo Alto Research Center, Palo Alto, Calif., (1979).
23. Stallman, R. M. *EMACS Manual for ITS Users.* AI Lab Memo 554, MIT, Cambridge, Mass., (1980).
24. Stanford Center for Information Processing. *Wylbur/370 The Stanford Timesharing System Reference Manual, 3rd ed.* Stanford University, Stanford, Calif., (1975).
25. Tesler, L. The Smalltalk environment. *Byte 6*, 8, (Aug. 1981) 90-147. (There is no available GYPSY documentation. This paper describes the Smalltalk editor, which is based on many of the same design ideas as GYPSY.)
26. Wang Laboratories, Inc. *Wang Word Processor Operator's Guide, 3rd release.* Lowell, Mass., (1978).
27. Xerox Corporation. *8010 Star Information System Reference Guide.* Dallas, Texas, (1981). (See also [16].)

CR Categories and Subject Descriptors: H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors*; I.7.1 [**Text Processing:**] Text Editing—*languages*; I.7.2 [**Text Processing**]: Document Preparation—*languages*

General Terms: Experimentation, Human Factors.

Aditional Key Words and Phrases: human-computer interface, human-computer interaction, user model, user performance, user psychology, ergonomics, human factors, system design, system evaluation, text editing.

Received 3/82; revised and accepted 1/83