Exhibit 1014 – Part 6

replication is necessary to validate whether the results are consistent when users must remember and type commands. Factors influencing the results may be the relative number of commands and objects and the familiarity the user has with each.

Finally, pilot studies by James Foley at George Washington University suggest that object first may be more appropriate when using selection by pointing on graphic displays is used. Different thinking patterns may be engaged when users are faced with visually oriented interfaces (right brain) and when they use syntax-oriented command notations (left brain). The object-first approach also fits conveniently with the strategy of leaving an object selected (and highlighted) after an action is complete, so that, if the same object is used in the next action, it is already selected.

4.4.2 Symbols versus keywords

Further evidence that command structure affects performance comes from a comparison of 15 commands in a commercially used symbol-oriented text editor, and revised commands that had a more keyword-oriented style (Ledgard et al., 1980). Here are three sample commands:

Symbol editor	**Keyword editor**
`FIND:/TOOTH/;-1`	`BACKWARD TO "TOOTH"`
`LIST;10`	`LIST 10 LINES`
`RS:/KO/,/OK/;*`	`CHANGE ALL "KO" TO "OK"`

The revised commands performed the same functions. Single-letter abbreviations (`L;10` or `L 10 L`) were permitted in both editors, so the number of keystrokes was approximately the same. The difference in the revised commands was that keywords were used in an intuitively meaningful way, but there were no standard rules of formation. Eight subjects at three levels of text-editor experience used both versions in this counterbalanced order within-subjects design.

The results (Table 4.1) clearly favored the keyword editor, indicating that command-formation rules do make a difference. Unfortunately, no specific guidelines emerged except that we should avoid using unfamiliar symbols for new users of a given text editor, even if the users are experienced with other text editors. It is interesting that the difference in percentage of task completed between the symbol and keyword editor was small for the experienced users. One conjecture, supported in other studies, is that experienced computer users develop skill in dealing with strange notations and therefore are less affected by syntactic variations.

Table 4.1

Impact of revised text-editor commands on three levels of users. (Source: Ledgard et al., 1980.)

	Percentage of Task Completed		Percentage of Erroneous Commands	
Users	**Symbol**	**Keyword**	**Symbol**	**Keyword**
Inexperienced	28	42	19	11
Familiar	43	62	18	6.4
Experienced	74	84	9.9	5.6

4.4.3 Hierarchical structure and congruence

Carroll (1982) altered two design variables to produce four versions of a 16-command language for controlling a robot (Table 4.2). Commands could be hierarchical (verb–object–qualifier) or nonhierarchical (verb only) and congruent (for example, `ADVANCE/RETREAT` or `RIGHT/LEFT`) or noncongruent (`GO/BACK` or `TURN/LEFT`). Carroll uses *congruent* to refer to meaningful pairs of opposites (*symmetry* might be a better term). Hierarchical structure and congruence have been shown to be advantageous in psycholinguistic experiments. Thirty-two undergraduate subjects studied one of the four command sets in a written manual, gave subjective ratings, and then carried out paper-and-pencil tasks.

Subjective ratings prior to performing tasks showed that subjects disapproved of the nonhierarchical noncongruent form, and gave the highest rating for the nonhierarchical congruent form. Memory and problem-solving tasks showed that congruent forms were clearly superior, and that the hierarchical forms were superior for several dependent measures. Error rates were dramatically lower for the congruent hierarchical forms.

This study assessed performance of new users of a small command language. Congruence helped subjects to remember the natural pairs of concepts and terms. The hierarchical structure enabled subjects to master 16 commands with only one rule of formation and 12 keywords. With a larger command set—say, 60 or 160 commands—the advantage of hierarchical structure should increase, assuming that a hierarchical structure could be found to accommodate the full set of commands. Another conjecture is that retention should be facilitated by the hierarchical structure and congruence.

Carroll's study was conducted during a half-day period; with 1 week of regular use, differences probably would be reduced substantially. However, with intermittent users or with users under stress, the hierarchical congruent form might again prove superior. An online experiment might have been more realistic and would have brought out differences in command length

Table 4.2

Command sets and partial results. (Source: Carroll 1982.)

Congruent		Noncongruent	
Hierachical	**Non-hierarchical**	**Hierachical**	**Non-hierarchical**
MOVE ROBOT FORWARD	ADVANCE	MOVE ROBOT FORWARD	GO
MOVE ROBOT BACKWARD	RETREAT	CHANGE ROBOT BACKWARD	BACK
MOVE ROBOT RIGHT	RIGHT	CHANGE ROBOT RIGHT	TURN
MOVE ROBOT LEFT	LEFT	MOVE ROBOT LEFT	LEFT
MOVE ROBOT UP	STRAIGHTEN	CHANGE ROBOT UP	UP
MOVE ROBOT DOWN	BEND	MOVE ROBOT DOWN	BEND
MOVE ARM FORWARD	PUSH	CHANGE ARM FORWARD	POKE
MOVE ARM BACKWARD	PULL	MOVE ARM BACKWARD	PULL
MOVE ARM RIGHT	SWING OUT	CHANGE ARM RIGHT	PIVOT
MOVE ARM LEFT	SWING IN	MOVE ARM LEFT	SWEEP
MOVE ARM UP	RAISE	MOVE ARM UP	REACH
MOVE ARM DOWN	LOWER	CHANGE ARM DOWN	DOWN
CHANGE ARM OPEN	RELEASE	CHANGE ARM OPEN	UNHOOK
CHANGE ARM CLOSE	TAKE	MOVE ARM CLOSE	GRAB
CHANGE ARM RIGHT	SCREW	MOVE ARM RIGHT	SCREW
CHANGE ARM LEFT	UNSCREW	CHANGE ARM LEFT	TWIST
Subjective Ratings (1 = Best, 5 = Worst)			
	1.86	1.63	1.81	2.73
Test 1	14.88	14.63	7.25	11.00
Problem 1 errors	0.50	2.13	4.25	1.63
Problem 1 omissions	2.00	2.50	4.75	4.15

that would have been a disadvantage to the hierarchical forms because of the greater number of keystrokes required. However, the hierarchical forms could all be replaced with three-letter abbreviations (for example, `MAL` for `MOVE ARM LEFT`), thereby providing an advantage even in keystroke counts.

4.4.4 Consistency, congruence, and mnemonicity

An elegant demonstration of the importance of structuring principles comes from a study of four command languages for text editing (Green and Payne, 1984). Language L4 (Figure 4.6) is a subset of the commercial word processor based on the EMACS editor, but it uses several conflicting organizing principles. The authors simplified language L3 by using only the CTRL key, and using congruence and mnemonic naming where possible. Language L2 uses `CTRL` to mean *forward* and `META` to mean *backward*, but mnemonicity is sacrificed. Language L1 uses the same meaningful structure for `CTRL` and `META`, congruent pairs, and mnemonicity.

	L1	L2	L3	L4
move pointer forward a paragraph	CTRL-[	CTRL-A	CTRL-]	META-]
move pointer backward a paragraph	META-[	META-A	CTRL-[	META-[
move pointer forward a sentence	CTRL-S	CTRL-B	CTRL-)	META-E
move pointer backward a sentence	META-S	META-B	CTRL-(	META-A
view next screen	CTRL-V	CTRL-C	CTRL-V	CTRL-V
view previous screen	META-V	META-C	CTRL-~^	META-V
move pointer to next line	CTRL-<	CTRL-D	CTRL-N	CTRL-N
move pointer to previous line	META-<	META-D	CTRL-P	CTRL-P
move pointer forward a word	CTRL-W	CTRL-E	CTRL-}	META-F
move pointer backward a word	META-W	META-E	CTRL-{	META-B
redisplay screen	CTRL-R	CTRL-F	CTRL-Y	CTRL-L
undo last command	META-G	META-G	CTRL-U	CTRL-G
kill sentence forward	CTRL-Z	CTRL-H	CTRL-S	META-K
kill line	CTRL-K	CTRL-I	CTRL-K	CTRL-K
delete character forward	CTRL-D	CTRL-J	CTRL-D	CTRL-D
delete character backward	META-D	META-J	CTRL-DEL	CTRL-DEL
delete word forward	CTRL-DEL	CTRL-K	CTRL-X	META-D
delete word backward	META-D	META-K	CTRL-W	META-DEL
move pointer forward a character	CTRL-C	CTRL-L	CTRL-F	CTRL-F
move pointer backward a character	META-C	META-L	CTRL-B	CTRL-B
move pointer to end of file	CTRL-F	CTRL-M	CTRL-~>	META-~>
move pointer to beginning of file	META-F	META-M	CTRL-~<	META-~<
move pointer to end of line	CTRL-L	CTRL-N	CTRL-Z	CTRL-E
move pointer to beginning of line	META-L	META-N	CTRL-A	CTRL-A
forward string search	CTRL-X	CTRL-O	CTRL-S	CTRL-S
reverse string search	META-X	META-O	CTRL-R	CTRL-R

Figure 4.6

The four languages used in the study discussed in the text. (Green, T. R. G. and Payne, S. J., "Organization and learnability in computer languages," *International Journal of Man–Machine Studies* 21 (1984) 7–18. Used by permission of Academic Press Inc. [London] Limited.)

Forty undergraduate subjects with no word-processing experience were given 12 minutes to study one of the four languages (Figure 4.6). Then, they were asked to recall and write on paper as many of the commands as possible. This step was followed by presentation of the command descriptions, after which they were asked to write down the associated command syntax. The free-recall and prompted-recall tasks were both repeated. The results showed a statistically significant difference for languages, with subjects using L4 demonstrating the worst performance. The best performance was attained with L1, which has the most structure. An online test would have been a useful followup to demonstrate the advantage obtained with practice and over a longer period.

In summary, sources of structure that have proven advantageous include these:

- Positional consistency
- Grammatical consistency
- Congruent pairing
- Hierarchical form

In addition, as discussed in the next section, a mixture of meaningfulness, mnemonicity, and distinctiveness is helpful.

One remaining form of structure is *visual* or *perceptual form*. The up- and down-arrows are highly suggestive of function, as are characters such as right- and left-angle brackets, the plus sign, and the ampersand. WORDSTAR takes advantage of a perceptual clue embedded in the QWERTY keyboard layout:

```
  E
A S D F
  X
```

CTRL-E moves the cursor up one line, CTRL-X moves the cursor down one line, CTRL-S moves the cursor one character left, CTRL-D moves the cursor one character right, CTRL-A moves the cursor one word left, and CTRL-F moves the cursor one word right. Other word processors use a similar principle with the CTRL-W, A, S, and Z keys or the CTRL-I, J, K, and M keys.

4.5 Naming and Abbreviations

In discussing command-language names, Schneider (1984) takes a delightful quote from Shakespeare's *Romeo and Juliet*: "A rose by any other name would smell as sweet." As Schneider points out, the lively debates in design circles suggest that this concept does not apply to command-language names. Indeed, the command names are the most visible part of a system and are likely to provoke complaints from disgruntled users.

Critics (Norman, 1981, for example) focus on the strange names in UNIX, such as `MKDIR` (make directory), `CD` (change directory), `LS` (list directory), `RM` (remove file), and `PWD` (print working directory); or in IBM's CMS, such as `SO` (temporarily suspend recording of trace information), `LKED` (link edit), `NUCXMAP` (identify nucleus extensions), and `GENDIRT` (generate directory). Part of the concern is the inconsistent abbreviation strategies that may take the first few letters, first few consonants, first and last letter, or first letter of

each word in a phrase. Worse still are abbreviations with no perceivable pattern.

4.5.1 Specificity versus generality

Names are important for learning, problem solving, and retention over time. With only a few names, a command set is relatively easy to master; but with hundreds of names, the choice of meaningful, organized sets of names becomes more important. Similar results were found for programming tasks, where variable name choices were less important in small modules with from 10 to 20 names than in longer modules with dozens or hundreds of names.

In a word-processing training session (Landauer et al., 1983), 121 students learned one of three command sets containing only three commands: the old set (`delete, append, substitute`), a new supposedly improved set (`omit, add, change`), and a random set designed to be confusing (`allege, cipher, deliberate`). Task performance times were essentially the same across the three command sets, although subjective ratings indicated a preference for the old set. The random names were highly distinctive and the mismatch with function may have been so disconcerting as to become memorable. These results apply to only small command sets.

With larger command sets, the names do make a difference, especially if they support congruence or some other meaningful structure. One naming-rule debate revolves around the question of *specificity versus generality* (Rosenberg, 1982). Specific terms can be more descriptive, and if they are more distinctive, they may be more memorable. General terms may be more familiar and therefore easier to accept. Two weeks after a training session with 12 commands, subjects were more likely to recall and recognize the meaning of specific commands than of general commands (Barnard et al., 1982).

In a paper-and-pencil test, 84 subjects studied one of seven sets of eight commands (Black and Moran, 1982). Two of the eight commands—the commands for inserting and deleting text—are shown here in all seven versions:

Infrequent, discriminating words	`insert`	`delete`
Frequent, discriminating words	`add`	`remove`
Infrequent, nondiscriminating words	`amble`	`perceive`
Frequent, nondiscriminating words	`walk`	`view`
General words (frequent, nondiscriminating)	`alter`	`correct`
Nondiscriminating nonwords (nonsense)	`GAC`	`MIK`
Discriminating nonwords (icons)	`abc-adbc`	`abc-ac`

The "infrequent, discriminating" command set resulted in faster learning and superior recall than did other command sets. The general words were correlated with the lowest performance on all three measures. The nonsense words did surprisingly well, supporting the possibility that, with small command sets, distinctive names are helpful even if they are not meaningful.

4.5.2 Abbreviation strategies

Even though command names should be meaningful for human learning, problem solving, and retention, they must satisfy another important criterion. They must be in harmony with the mechanism for expressing the commands to the computer. The traditional and widely used command-entry mechanism is the keyboard, which indicates that commands should use brief and kinesthetically easy codes. Commands requiring shifted keys or CTRL keys, special characters, or difficult sequences are likely to cause higher error rates. For text editing, when many commands are applied and speed is appreciated, single-letter approaches are attractive. Overall, brevity is a worthy goal since it can speed entry and reduce error rates. Many word-processor designers have pursued this approach, even when mnemonicity was sacrificed, thereby making use more difficult for novice and intermittent users.

In less demanding applications, designers have used longer command abbreviations, hoping that the gains in recognizability would be appreciated over the reduction in key strokes. Novice users may actually prefer typing the full name of a command because they have a greater confidence in its success (Landauer et al., 1983). Novices who were required to use full command names before being taught two-letter abbreviations made fewer errors with the abbreviations than those who were taught the abbreviations from the start and than did those who could create their own abbreviations (Grudin and Barnard, 1985).

The phenomenon of preferring the full name at first appeared in our study of bibliographic retrieval with the Library of Congress's SCORPIO system. Novices preferred typing the full name, such as `BROWSE` or `SELECT`, rather than the traditional four-letter abbreviations `BRWS` or `SLCT`, or the single-letter abbreviations `B` or `S`. After five to seven uses of the command, their confidence increased and they attempted the single-letter abbreviations. A designer of a text adventure game recognized this principle and instructs novice users to type `EAST`, `WEST`, `NORTH`, or `SOUTH`; after five full-length commands, the system tells the user about the single-letter abbreviations. A related report comes from some users of IBM's CMS, who find that the minimal length abbreviations are too difficult to learn; they stick with the full form of the command.

With experience and frequent use, abbreviations become attractive for, and even necessary to satisfy, the "power" user. Efforts have been made to find optimal abbreviation strategies. Several studies support the notion that abbreviation should be made by a consistent strategy (Ehrenreich and Porcu, 1982; Benbasat and Wand, 1984; Schneider, 1984). Here are six potential strategies:

1. *Simple truncation*: Use the first, second, third, etc. letters of each command. This strategy requires that each command be distinguishable by the leading string of characters. Abbreviations can be all of the same length or of different lengths.
2. *Vowel drop with simple truncation*: Eliminate vowels and use some of what remains. If the first letter is a vowel, it may or may not be retained. H, Y, and W may or may not be considered as vowels.
3. *First and last letter*: Since the first and last letters are highly visible, use them; for example, use `ST` for `SORT`.
4. *First letter of each word in a phrase*: Use this popular technique, for example, with a hierarchical design plan.
5. *Standard abbreviations from other contexts*: Use familiar abbreviations such as `QTY` for `QUANTITY`, `XTALK` for `CROSSTALK` (a software package), `PRT` for `PRINT`, or `BAK` for `BACKUP`.
6. *Phonics*: Focus attention on the sound; for example, use `XQT` for `execute`.

Truncation appears to be the most effective mechanism overall, but it has its problems. Conflicting abbreviations appear often, and decoding of an unfamiliar abbreviation is not as easy as when vowel dropping is used (Schneider, 1984).

4.5.3 Guidelines for using abbreviations

Ehrenreich and Porcu (1982) offer this compromise set of guidelines:

1. A *simple*, primary rule should be used to generate abbreviations for most items; a *simple* secondary rule should be used for those items where there is a conflict.
2. Abbreviations generated by the secondary rule should have a marker (for example, an asterisk) incorporated in them.
3. The number of words abbreviated by the secondary rule should be kept to a minimum.
4. Users should be familiar with the rules used to generate abbreviations.
5. Truncation is an easy rule for users to comprehend, but it may also produce a large number of identical abbreviations for different words.
6. Use fixed-length abbreviations in preference to variable-length ones.

7. Abbreviations should not be designed to incorporate endings (e.g., ING, ED, S).
8. Unless there is a critical space problem, abbreviations should not be used in messages generated by the computer and read by the user.

Abbreviations are an important part of system design and they are appreciated by experienced users. Users are more likely to use abbreviations if they are confident in their knowledge of the abbreviations and if the benefit is a savings of more than one to two characters (Benbasat and Wand, 1984). The appearance of new input devices and strategies (for example, selecting by pointing) will change the criteria for abbreviations. Each situation has its idiosyncrasies and should be evaluated carefully by the designer, applying empirical tests where necessary.

4.6 Command Menus

To relieve the burden of memorization of commands, some designers offer users brief prompts of available commands. The early online version of the *Official Airline Guide* used such prompts as this:

```
ENTER +,L#,X#,S#,R#,M,RF(#=LINE NUMBER)
```

This prompt reminds users of the commands related to fares that have been displayed, and the related flight schedules:

+	move forward one screen
L#	limitations on airfares
X#	detailed information on a listed flight
S#	schedule information for the listed fare
R#	return flight information for this route
M	main menu
RF	return fares

Experienced users come to know the commands and do not need to read the prompt or the help screens. Intermittent users know the concepts and refer to the prompt to jog their memory and to get help in retaining the syntax for future uses. Novice users do not benefit as much from the prompt and must take a training course or consult the online help.

The prompting approach emphasizes syntax and serves frequent users. It is closer to but more compact than a standard numbered menu, and preserves screen space for task-related information. The early WORDSTAR

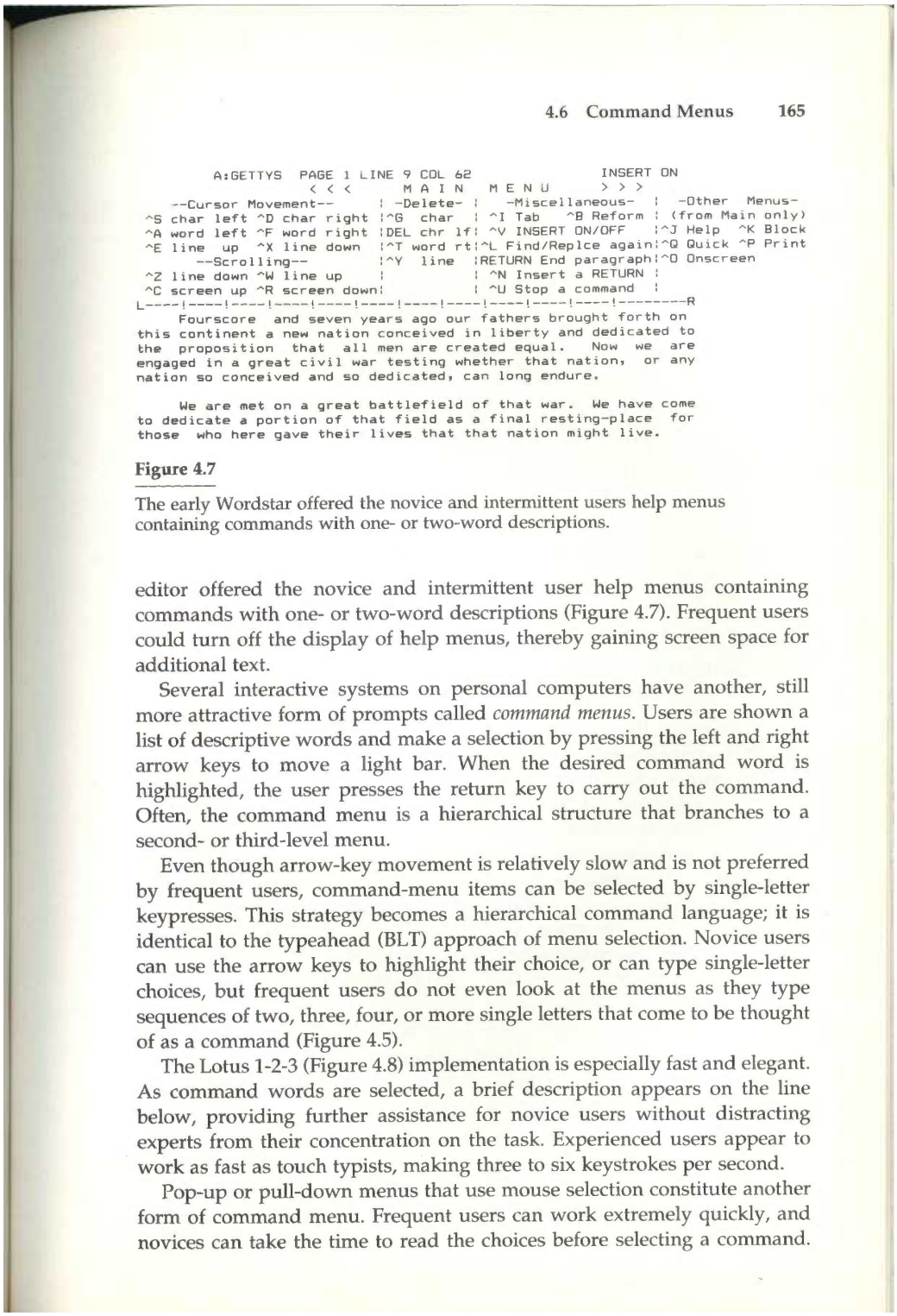

Figure 4.7

The early Wordstar offered the novice and intermittent users help menus containing commands with one- or two-word descriptions.

editor offered the novice and intermittent user help menus containing commands with one- or two-word descriptions (Figure 4.7). Frequent users could turn off the display of help menus, thereby gaining screen space for additional text.

Several interactive systems on personal computers have another, still more attractive form of prompts called *command menus*. Users are shown a list of descriptive words and make a selection by pressing the left and right arrow keys to move a light bar. When the desired command word is highlighted, the user presses the return key to carry out the command. Often, the command menu is a hierarchical structure that branches to a second- or third-level menu.

Even though arrow-key movement is relatively slow and is not preferred by frequent users, command-menu items can be selected by single-letter keypresses. This strategy becomes a hierarchical command language; it is identical to the typeahead (BLT) approach of menu selection. Novice users can use the arrow keys to highlight their choice, or can type single-letter choices, but frequent users do not even look at the menus as they type sequences of two, three, four, or more single letters that come to be thought of as a command (Figure 4.5).

The Lotus 1-2-3 (Figure 4.8) implementation is especially fast and elegant. As command words are selected, a brief description appears on the line below, providing further assistance for novice users without distracting experts from their concentration on the task. Experienced users appear to work as fast as touch typists, making three to six keystrokes per second.

Pop-up or pull-down menus that use mouse selection constitute another form of command menu. Frequent users can work extremely quickly, and novices can take the time to read the choices before selecting a command.

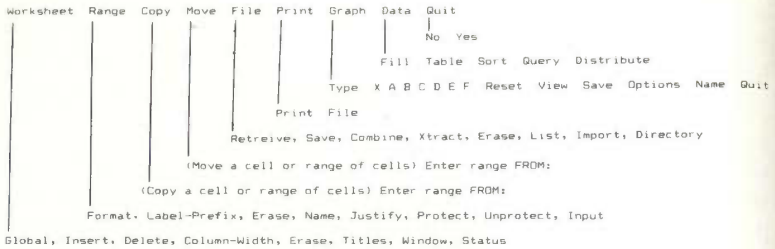

Figure 4.8

The first two levels of command menus from LOTUS 1-2-3 reveal the rich function available to users. At the third level, users may receive another menu or enter values. (Printed with permission of Lotus Development Corporation, Cambridge, MA.)

With a fast display, command menus blur the boundaries between commands and menus.

4.7 Natural Language in Computing

Even before there were computers, people dreamed about creating machines that would accept *natural language*. It is a wonderful fantasy, and the success of word-manipulation devices such as word processors, printing presses, tape recorders, and telephones may give encouragement to some people. Although there has been some progress in machine translation from one natural language to another (for example, Japanese to English), most effective systems require constrained or preprocessed input, or postprocessing of output. Undoubtedly, improvements will continue and constraints will be reduced, but high-quality reliable translations of complete documents without human intervention seems difficult to attain. Structured texts such as weather reports are translatable; technical papers are marginally translatable; novels or poems are not translatable. Language is subtle; there are many special cases, meaning is not easily programmed into machines, and context has a powerful and pervasive effect.

Although full comprehension and generation of language seems inaccessible, there are still many ways that computers can be used in dealing with natural language, such as for interaction, queries, database searching, text generation, and adventure games. So much research has been invested in natural-language systems that undoubtedly some successes will emerge, but widespread use may not develop because the alternatives may be more appealing. More rapid progress can be made if carefully controlled experi-

mental tests are used to discover the designs, users, and tasks for which natural-language applications are most beneficial.

4.7.1 Natural-language interaction

Researchers hope that someday computers will respond easily to commands users issue by typing or speaking in natural language. *Natural-language interaction* (NLI) might be defined as the operation of computers by people using a familiar natural language (such as English) to give instructions and receive responses. Users do not have to learn a command syntax or to select from menus. Early attempts at generalized "automatic programming" have faded, but there are continuing efforts to provide domain-specific assistance.

The problems with NLI lie in not only implementation on the computer, but also desirability for large numbers of users for a wide variety of tasks. People are different from computers, and human–human interaction is not necessarily an appropriate model for human operation of computers. Since computers can display information 1000 times faster than people can enter commands, it seems advantageous to use the computer to display large amounts of information, and to allow novice and intermittent users simply to choose among the items. Selection helps to guide the user by making clear what functions are available. For knowledgeable and frequent users, who are thoroughly aware of the available functions, a precise, concise command language is usually preferred.

In fact, the metaphors of artificial intelligence (smart machines, intelligent agents, and expert systems) may prove to be mind-limiting distractions that inhibit designers from creating the powerful tools that become commercially successful. Spreadsheets, WYSIWYG word processors, and direct-manipulation graphics tools emerged from a recognition of what users were using effectively, rather than from the misleading notions of intelligent machines. Similarly, the next generation of groupware to support collaboration, visualization and simulation packages, tele-operated devices, and hypermedia stem from user-centered scenarios, rather than the machine-centered artificial-intelligence fantasies.

The SSOA model can help us to sort out the issues. NLI does not provide information about actions and objects in the task domain; users are usually presented with a simple prompt that invites a natural-language query. But the user may be knowledgeable about the task domain—for example, about the meaning of database objects and permissible actions. NLI also does not necessarily convey knowledge of the computer concepts—for example, tree-structuring of information, implications of a deletion, Boolean operations, or query strategies. NLI does relieve the user of learning new syntactic rules, since it presumably will accept familiar English language requests. Therefore, NLI can be effective for the user who is knowledgeable about some task

domain and computer concepts but who is an intermittent user who cannot retain the syntactic details. Lately, some members of the artificial-intelligence community have understood the power of direct manipulation for conveying the system state and suggesting possible actions, and have attempted to blend visual presentation of status with natural-language input.

NLI might apply to checkbook maintenance (Shneiderman, 1980), where the users recognize that there is an ascending sequence of integer-numbered checks, and that each check has a single payee field, single amount, single date, and one or more signatures. Checks can be issued, voided, searched, and printed. In fact, following this suggestion, Ford (1981) created and tested an NLI system for this purpose. Subjects were paid to maintain their checkbook registers by computer using an APL-based program that was incrementally refined to account for unanticipated entries. The final system successfully handled 91 percent of users' requests, such as these:

```
Pay to Safeway on 3/24/86 $29.75.
June 10 $33.00 to Madonna.
Show me all the checks paid to George Bush.
Which checks were written on October 29?
```

Users reported satisfaction with the system and were eager to use the system even after completing the several months of experimentation. This study can be seen as a success for NLI, but alternatives might be even more attractive. Showing a full screen of checkbook entries with a blank line for new entries might accomplish most tasks without any commands and with minimal typing (similar to what is used in Quicken from Intuit). Users could do searches by entering partial information (for example, `George Bush` in the payee field) and then pressing a query key.

There have been numerous informal tests of NLI systems, but only a few have been experimental comparisons against some other design. Researchers seeking to demonstrate the advantage of NLI over command-language and menu approaches for creating business graphics were surprised to find no significant differences for time, errors, or attitude (Hauptmann and Green, 1983).

A more positive result was found with users of HAL, the restricted natural-language addition to Lotus 1-2-3 (Napier et al., 1989). HAL users could avoid the command-menu `/WEY` (for Worksheet Erase Yes), and could type requests such as `\erase worksheet`, `\insert row`, or `\total all columns`, starting with any of the 180 permissible verbs. In an empirical study, after 1.5 days of training, 19 HAL users and 22 Lotus 1-2-3 users worked on three substantial problem sets for another 1.5 days. Performance and preference strongly favored the restricted natural language version, but the experimenters had difficulty identifying the features that made a differ-

ence: "It is not clear whether Lotus HAL was better because it is more like English or because it takes advantage of context, but we suspect the latter is more important." By context, the authors meant features such as the cursor position or meaningful variable names that indicate cell ranges.

Some NLI work has turned to automatic speech recognition and speech generation to reduce the barriers to acceptance. There is some advantage to use of these technologies, but the results are still meager. A promising application is the selection of painting tools by discrete-word recognition (see Section 6.4.1), thus eliminating the frustration and delay of moving the cursor from the object to the tool menu on the border and back again (Pausch, 1991). Selections are voiced but feedback is visual. Users of the mouse plus the voice commands performed their tasks 21-percent faster than did the users who had only the mouse. Alternatives to voice, such as keyboard or touchscreen, were not tested.

There is some portion of the user spectrum that can benefit from NLI, but it may not be as large as promoters believe. Computer users usually seek predictable responses and are discouraged if they must engage in clarification dialogs frequently. Since NLI has such varied forms, the users must constantly be aware of the computer's response, to verify that their actions were recognized. Finally, visually oriented interactions, embracing the notions of direct manipulation (see Chapter 5), make more effective use of the computer's capacity for rapid display. In short, pointing and selecting in context is often more attractive than typing or even speaking an English sentence.

It is surprising that designers of expert systems have attempted to embed NLI. Expert systems already tax the user with complexity, lack of visibility of the underlying processes, and confusion about what functions the system can and cannot handle. A precise, concise notation or selection in context from a list of alternatives seems far more suitable in providing users with predictable and comprehensible behavior (Hayes-Roth, 1984) (Figure 4.9).

4.7.2 Natural-language queries

Since general interaction is difficult to support, some designers have pursued a more limited goal of *natural-language queries* (NLQ) against relational databases. The *relational schema* contains attribute names, and the database contains attribute values, both of which are helpful in disambiguating queries. A simulated query system was used to compare a subset of the structured SQL database facility to a natural-language system (Small and Weldon, 1983). The SQL simulation resulted in faster performance on a benchmark set of tasks. Similarly, a field trial with a real system, users, and queries pointed to the advantages of SQL over the natural-language alternative (Jarke et al., 1985). Believers in NLQ may claim that more research and system development is

Sample Session from an Expert System for Oil Drilling Advisor

What is the name of WELL–159?
AGF7–93E

What is the profile of AGF7–93E?
DEVIATED

Please enter information about FORMATION–1:

upper-limit meters	lower-limit meters	main-rock-type	homogenous/ interbedded
747	806	SHALE	HOMOGENOUS

Please enter information on PROBLEM–1:

problem-type	prior-action	total-depth	casting-shoe depth
STICKING	REAMING	1111 METERS	747 METERS

Please enter the composition of the drill-string starting from the bit (type ? for assistance):?
BIT 9"5/8 STAPB"5/8 SHORTDC7"3/4STA9"5/8...NDP5

What was the drilling method employed when the problem occurred:
ROTARY

What is the depth of the freepoint?
UNKNOWN

Figure 4.9

This extract demonstrates one designer's attempt at an expert-system dialog. User input is shown in upper-case letters. Users must type in values even when selection from a menu would be more meaningful, more rapid, and less error-prone. Furthermore, there does not appear to be any way to go back to change values, to view values, or to reuse values from previous sessions. (F. Hayes-Roth, The knowledge-based expert system: A tutorial, *IEEE Computer 17*, 9 (September 1984), 11–28. Copyright 1984 IEEE.)

needed before NLQ, can be excluded, but improvements in menus, command languages, and direct manipulation seem equally likely.

Supporters of NLQ can point with some pride at the modest success of the commercially available INTELLECT system, which has approximately 400 installations on large mainframe computers (Figure 4.10); the system's appeal has faded in recent years, however, as users turn to other approaches. Business executives, sales representatives, and other people use INTELLECT to search databases on a regular basis. Several innovative implementation ideas help to make INTELLECT successful. First, the parser uses the contents of the database to parse queries; for example, the parser can determine that a query containing `Cleveland` refers to city locations, because `Cleveland` is an instance in the database. Second, the system

```
PLEASE ENTER YOUR FIRST REQUEST

WHAT'S IN THE DATABASE?
FIELDS IN THE FILE OF EMPLOYEES

NAME                    JOB          SALARY          SEX
AGE                     FAMILY       CITY            STATE
EMPLOYEE NUMBER

WHO WORKS IN NEW YORK CITY
PRINT THE JOB AND NAME OF ALL EMPLOYEES WITH CITY = NEW YORK

PROFESSIONAL                        NAME
OCCUPATION

MACHINIST                           ANGELIN
PHYSICIAN                           ANGUS

WHAT IS THE NUMBER OF ANGUS
COUNT THE EMPLOYEES WITH NAME = ANGUS

ANSWER:  1

WHAT IS THE EMPLOYEE NUMBER OF ALL EMPLOYEES WITH NAME = ANGUS
EMPLOYEE
NUMBER

0034

WHICH WOMEN WORK IN NEW YORK CITY
PRINT THE JOB OF ALL EMPLOYEES WITH SEX = FEMALE & CITY = NEW YORK

PROFESSIONAL
OCCUPATION

MACHINIST
ADMINISTRATOR
ELECTRICIAN

FEMALE EMPLOYEES IN NEW YORK CITY
PRINT THE NAME OF ALL EMPLOYEES WITH SEX = FEMALE & CITY = NEW YORK

NAME

ANGELIN
CANTEBURY
EDWARDS

HOW MANY NEW YORK CITY MEN ARE OVER 30 YEARS OLD
COUNT THE EMPLOYEES WITH CITY = NEW YORK & SEX = MALE & AGE>30

ANSWER:  9

PRINT THEM
PRINT THE NAME AND AGE OF ALL EMPLOYEES WITH CITY = NEW YORK &
     SEX = MALE & AGE>30

                YEARS
NAME              OF
                 AGE

ANGUS             43
ELEY              69
HILTON            44
```

Figure 4.10

Demonstration session with Intellect. User input is underscored. Intellect rephrases user input into a structured query language, which users often mimic as they become more knowledgeable. (AI Corp., Cambridge, MA.)

administrator can conveniently include guidance for handling domain-specific requests, by indicating fields related to who, what, where, when, how, etc. queries. Third, INTELLECT rephrases the user's query and displays a response, such as `PRINT THE CHECK NUMBERS WITH PAYEE = GEORGE BUSH`. This structured response serves as an educational aid, and users gravitate toward expressions that mimic the style. Eventually, as

users become more knowledgeable, they often use concise, commandlike expressions that they believe will be parsed successfully. Even the promoters of INTELLECT recognize that novice users who are unfamiliar with the task domain will have a difficult time, and that the ideal user is a knowledgeable intermittent user.

A more successful product is Q & A from Symantec, which provides rapid, effective query interpretation and execution on IBM PCs (Figure 4.11). The package makes a very positive impression, but few data have been collected about actual usage. The designers cite many instances of happy NLQ users and find practical applications in their daily work, but the popularity of the package seems to be more closely tied to the fine word processor plus database, and the form fillin facilities. A further NLQ package is CLOUT, which is part of the Rbase package.

An innovative blend of NLQ and menus was developed under the name NLMENU (Tennant et al., 1983) and is distributed by Texas Instruments under the name NaturalLink. Natural-language phrases are shown as a series of menus. As phrases (for example, `FIND/COLOR/AND/NAME OF PARTS/WHOSE COLOR IS`) are chosen by a pointing strategy, a query is formed in a command window. Users receive information from the menus, obviating the need for a query. For example, if the database of parts and suppliers contains only red, green, and blue parts, only these choices appear in the window containing the `PART COLOR` menu. Users can see the full range of possible queries and thereby can avoid the frustration of probing the boundaries of system functionality. With this strategy, typing is eliminated, and the user is guaranteed a semantically and syntactically correct query.

4.7.3 Text-database searching

Searching of textual databases is a growing application for natural-language enthusiasts who have developed filters and parsers for queries expressed in natural language. For example, in a legal application (`Find cases of tenants who have sued landlords for lack of heat`) the filter eliminates pronouns and other noise words, provides synonyms from a thesaurus (`renters`), deals with singulars versus plurals, and handles other problems such as misspellings or foreign terms. Then, the parser separates the query into standard components, such as `plaintiff`, `defendant`, and `cause`; the query can now be used to search a more structured database.

Another application with textual databases is to use natural-language parsing strategies to parse the stored text and to arrange it into more structured formats. The advantage here is that the parsing can be done once in advance to structure the entire database and to speed searches when users pose queries. Legal (Supreme Court decisions or state laws), medical (scientific journal articles or patient histories), and journalistic (Associated

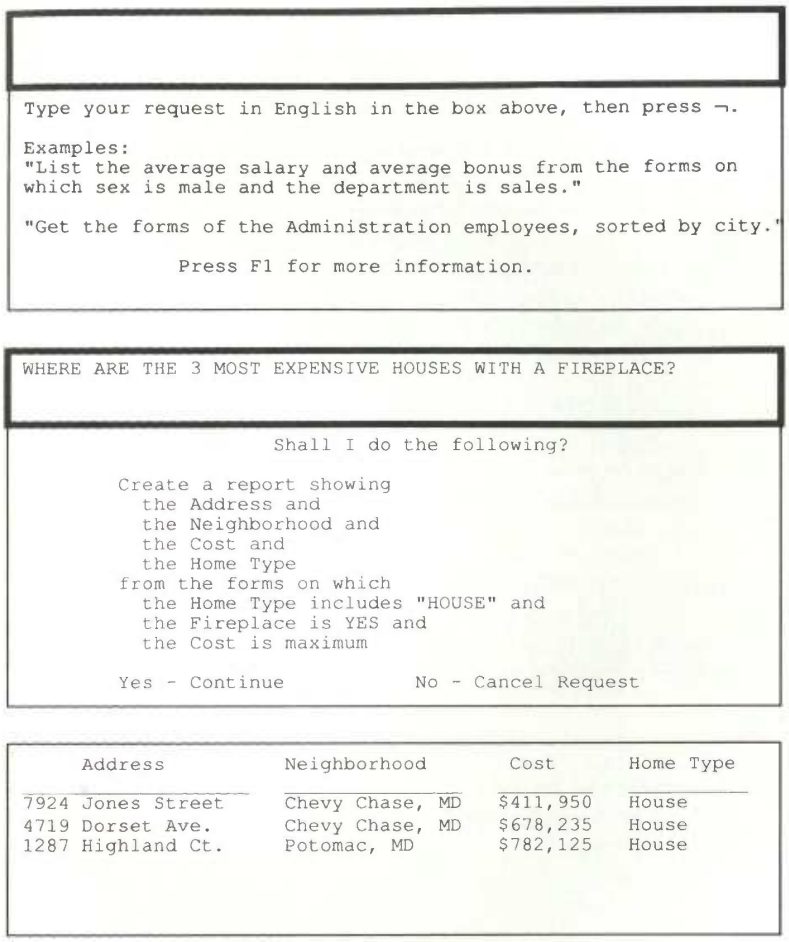

Figure 4.11

Q&A supports a natural language front end for its database. Users can type questions in English and produce database search results. This figure shows the three stages: forming a query, verifying it, and receiving the results. (Q&A is a product of Symantec Corp., Cupertino, CA.)

Press news stories or Dow Jones reports) texts have been used. This application is promising because even a modest increase in suitable retrievals is appreciated by users, and incorrect retrievals are tolerated better than are errors in NLI.

4.7.4 Natural-language text generation

Although the artificial-intelligence community often frowns on *natural-language text generation* (NLTG) as a modest application, it does seem to be a worthy one (Fedder, 1990). It handles certain simple tasks, such as the preparation of structured weather reports (`80 percent chance of light rain in northern suburbs by late Sunday afternoon`) from complex mathematical models. These reports can be sent out automatically, or even can be used to generate spoken reports available over the telephone. More elaborate applications of NLTG include preparation of reports of medical laboratory or psychological tests. The computer generates not only readable reports (`White-blood-cell count is 12,000`), but also warnings (`This value exceeds the normal range of 3000 to 8000 by 50 percent`) or recommendations (`Further examination for systemic infection is recommended`). Still more involved NLTG scenarios involve the creation of legal wills, contracts, or business proposals.

On the artistic side, computer generation of poems and even novels is a regular discussion point in literary circles. Although computer-generated combinations of randomly selected phrases can be provocative, it is still the creative work of the person who chose the set of possible words and decided which of the outputs to publish. This position parallels the customary attitude of crediting the human photographer, rather than the camera or the subject matter of the photograph.

4.7.5 Adventure and educational games

A notable and widespread success of NLI techniques is in the variety of adventure games (Figure 4.12). Users may indicate directions of movement or type commands, such as `TAKE ALL OF THE KEYS`, `OPEN THE GATE`, or `DROP THE CAGE AND PICK UP THE SWORD`. Part of the attraction of NLI in this situation is that the system is unpredictable, and some exploration is necessary to discover the proper incantation.

4.8 Practitioner's Summary

Command languages can be attractive when frequent use of a system is anticipated, users are knowledgeable about the task domain and computer concepts, screen space is at a premium, response time and display rates are slow, and numerous functions that can be combined in many ways are supported. Users have to learn the semantics and syntax, but they can initiate rather than respond, rapidly specifying actions involving several

Figure 4.12

This adventure game is modeled on the Wizard of Oz story. The user types phrases such as `open the door` or `take slippers` or abbreviations such as `s` to move south. More complex phrases, such as `put the hat on the scarecrow`, are possible. (Courtesy of Spinnaker Software, Cambridge, MA.)

objects and options. Finally, complex sequences of commands can be easily specified and stored for future use as a macro.

Designers should begin with a careful task analysis to determine what functions should be provided. Hierarchical strategies and congruent structures facilitate learning, problem solving, and human retention over time. Laying out the full set of commands on a single sheet of paper helps to show the structure to the designer and to the learner. Meaningful specific names aid learning and retention. Compact abbreviations constructed according to a consistent rule facilitate retention and rapid performance for frequent users.

Innovative strategies, such as command menus, can be effective if rapid response to screen actions can be provided. Natural-language interaction can be implemented partially, but its advantage for widespread application has yet to be demonstrated.

4.9 Researcher's Agenda

Designers could be helped by development of strategies for task analysis, taxonomies of command-language designs, and criteria for using commands or other techniques. The benefits of structuring command languages based on such concepts as hierarchical structure, congruence, consistency, and mnemonicity have been demonstrated in specific cases, but replication in varied situations is important. Experimental testing should lead to a more comprehensive cognitive model of command language learning and use (Table 4.3).

A generator of command-language systems would be a useful tool for research and development of new command languages. The designer could provide a formal specification of the command language, and the system

Table 4.3

Command Language Guidelines

- Create an explicit model of objects and actions.
- Choose meaningful, specific, distinctive names.
- Implement a hierarchical structure where possible.
- Provide a consistent structure (hierarchy, argument order, action–object).
- Support consistent abbreviation rules (preferably truncation to one letter).
- Offer frequent users the capability to create macros.
- Use command menus on high-speed displays when appropriate.
- Limit the number of commands and the ways of accomplishing a task.

would generate an interpreter. With experience in using such a tool, design analyzers might be built to critique the design, to detect ambiguity, to check for consistency, to verify completeness, to predict error rates, or to suggest improvements. Even a simple but thorough checklist for command-language designers would be a useful contribution.

Novel input devices and high-speed, high-resolution displays offer new opportunities, such as command and pop-up menus, for breaking free from the traditional syntax of command languages. Natural-language interaction still holds promise in certain applications, and empirical tests offer us a good chance to identify the appropriate niches and design strategies.

References

Barnard, P. J. and Hammond, N. V., Cognitive contexts and interactive communication, IBM Hursley (U.K.) Human Factors Laboratory Report HF070 (December 1982).

Barnard, P., Hammond, N., MacLean, A., and Morton, J., Learning and remembering interactive commands, *Proc. Conference on Human Factors in Computer Systems*, available from ACM, Washington DC (1982), 2–7.

Barnard, P. J., Hammond, N. V., Morton, J., Long, J. B., and Clark, I. A., Consistency and compatibility in human–computer dialogue, *International Journal of Man–Machine Studies 15* (1981), 87–134.

Benbasat, Izak and Wand, Yair, Command abbreviation behavior in human–computer interaction, *Communications of the ACM 27*, 4 (April 1984), 376–383.

Black, J., and Moran, T., Learning and remembering command names, *Proc. Conference on Human Factors in Computer Systems*, available from ACM Washington, DC (1982), 8–11.

Carroll, John M., Learning, using and designing command paradigms, *Human Learning 1*, 1 (1982), 31–62.

Carroll, J. M., and Thomas, J., Metaphor and the cognitive representation of computing systems, *IEEE Transactions on Systems, Man, and Cybernetics, SMC-12*, 2 (March/April 1982), 107–115.

Ehrenreich, S. L., and Porcu, Theodora, Abbreviations for automated systems: Teaching operators and rules. In Badre, Al, and Shneiderman, Ben, (Editors), *Directions in Human–Computer Interaction*, Ablex, Norwood, NJ (1982), 111–136.

Fedder, Lee., Recent approaches to natural language generation. In Diaper, D., Gilmore, D., Cockton, G., and Shackel, B. (Editors), *Human–Computer Interaction: Interact '90*, North-Holland, Amsterdam, The Netherlands (1990), 801–805.

Ford, W. Randolph, *Natural Language Processing by Computer — A New Approach*, Ph. D. Dissertation, Department of Psychology, Johns Hopkins University, Baltimore, MD (1981).

Green, T. R. G. and Payne, S. J., Organization and learnability in computer languages, *International Journal of Man–Machine Studies 21* (1984), 7–18.