Exhibit 1015 – Part 3

was then highlighted by reverse video until the RETURN key was pressed. Each subject worked for three minutes with each form. The results were:

	Mean tasks completed	Total errors	Preferred form
Item-RETURN	14.8	4	0 subjects
Immediate response	15.5	7	7 subjects
Highlight-RETURN	15.3	3	29 subjects

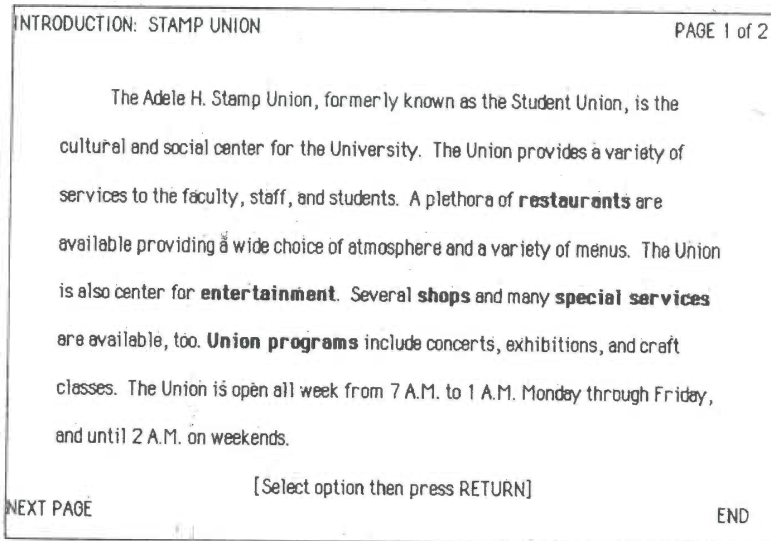
Subjects worked slightly faster with the immediate response form, but the error rate was highest. The subjective preference strongly favored the highlighting, which looks very impressive to novice users.

3.8 EMBEDDED MENUS

All the menus discussed thus far might be characterized as explicit menus in that there is an orderly enumeration of the menu items with little extraneous information. However, in many situations the menu items might be embedded in text or graphics and still be selectable.

In designing a textual database about people, events, and places for a museum application, it seemed natural to allow users to retrieve detailed information by selecting a name in context. Selectable names were highlighted and the user could move a reverse video bar among highlighted names by pressing the four arrow keys (Figure 3.10). Selection was made by pressing ENTER and the user obtained a new article plus the option of returning to the previous article. The names, places, phrases, or foreign language words were menu items embedded in meaningful text that informed the user and helped clarify the meaning of the items. Subsequent implementations used mouse selection or touchscreens, leading to the generic term *touchtext* for this application of embedded menus.

Touchtext was also used in an implementation of online maintenance manuals that provided diagnostic information in textual form on one


INTRODUCTION: STAMP UNION PAGE 1 of 2

The Adele H. Stamp Union, formerly known as the Student Union, is the cultural and social center for the University. The Union provides a variety of services to the faculty, staff, and students. A plethora of **restaurants** are available providing a wide choice of atmosphere and a variety of menus. The Union is also center for **entertainment**. Several **shops** and many **special services** are available, too. **Union programs** include concerts, exhibitions, and craft classes. The Union is open all week from 7 A.M. to 1 A.M. Monday through Friday, and until 2 A.M. on weekends.

[Select option then press RETURN]

NEXT PAGE END

Figure 3.10: Display from the database on the Adele H. Stamp Student Union at the University of Maryland showing the embedded menu style of TIES. A reverse video selector box initially covers the NEXT PAGE command. Users move the selector box over highlighted references or commands and then select by pressing RETURN. A touch screen version allows selection by merely touching the highlighted reference or command.

monitor and graphics assistance on a second monitor (Koved & Shneiderman, 1986).

Embedded menus have emerged in other applications. Air traffic control systems allow selection of airplanes in the spatial layout of flight paths to provide more detailed information for controllers. Geographic display systems allow selection of cities or zooming in on specific regions to obtain more information (Herot, 1984). In these applications, the items are icons, text, or regions in a two-dimensional layout.

Language-directed editors permit users to select programming language constructs for expansion during the program composition process (Teitelbaum & Reps, 1981). In a program browser at the University of Maryland, Phil Shafer offered programmers the capability of moving the

SKYHAWKE Ex. 1015, page 86

cursor onto a variable name or procedure invocation, pressing a function key, and receiving the data declaration or procedure definition in a separate window. The variable and procedure names were menu items embedded in the context of a Pascal program.

Many spelling checkers use the embedded menu concept by highlighting the possibly misspelled words in the context of their use. The author of the text can move a cursor to a highlighted word and request possible words or type in the correctly spelled word.

Embedded menus permit items to be viewed in context and they eliminate the need for a distracting and screen-wasting enumeration of items. Contextual display helps keep the users focused on their tasks and the objects of interest. Items rewritten in list form may require longer descriptions (of the items) and increase the difficulty of making selections because of confusion arising from cross-referencing between the menu and the context.

3.9 FORM FILL-IN

Menu selection is effective in choosing an item from a list, but some tasks are cumbersome with menus. If data entry of personal names or numeric values is required, then keyboard typing becomes more attractive. The keyboard may be viewed as a continuous single menu from which multiple selections are made rapidly. When many fields of data are necessary, the appropriate interaction style might be called form fill-in. For example, the user might be presented with a purchase order form for ordering from a catalog, as in Figure 3.11. Another example of a form using color coding is in Color Plate 1.

The form fill-in approach is attractive because the full complement of information is visible, giving the users a feeling of being in control of the dialog. Few instructions are necessary since this approach resembles familiar paper forms. On the other hand, users must be familiar with keyboards, use of the TAB key to move the cursor, error correction by backspacing, field label meanings, permissible field contents, and use of the ENTER key. Form fill-in must be done on displays, not hardcopy devices, and the display device must support cursor movement.

SKYHAWKE Ex. 1015, page 87

```
Type in the information below,
pressing TAB to move the cursor, and
press ENTER when done.

Name: ______________________  Phone: (___) ___-___

Address: ________________________

       : ________________________

City: ________________  State: __  Zip Code: ____

Charge Number: ___ ___ ___ ___

Catalog                Catalog
Number     Quantity    Number     Quantity

______     _____       ____       ____

______     _____       ____       ____

______     _____       ____       ____

______     _____       ____       ____
```

Figure 3.11: A form fill-in design for a department store.

An experimental comparison of database update by form fill-in and a command language strategy demonstrated a significant speed advantage for the form fill-in style (Ogden & Boyle, 1982). Eleven of the twelve subjects expressed a preference for the form fill-in approach.

3.9.1 Form fill-in design guidelines

There is a paucity of empirical work on form fill-in, but a number of design guidelines have emerged from practitioners (Galitz, 1980; Pakin & Wray, 1982; Brown, 1986). Many companies offer form fill-in creation

FORM FILL-IN GUIDELINES

- Meaningful title
- Comprehensible instructions
- Logical grouping and sequencing of fields
- Visually appealing layout
- Familiar field labels
- Consistent terminology and abbreviations
- Error correction for characters and fields
- Visual templates for common fields
- Help facilities

Table 3.3: Form fill-in guidelines based on practical experience, but in need of validation and clarification.

tools, such as Hewlett-Packard's Forms 3000, IBM's ISPF, Digital Equipment Corporation's FORM, Ashton-Tate's dBASE and Lotus Development Corporation's Symphony. Software tools simplify design, help ensure consistency, ease maintenance, and speed implementation. But even with excellent tools, the designer must still make many complex decisions (Table 3.3).

The elements of form fill-in design include:

- *Meaningful title*: Identify the topic and avoid computer terminology
- *Comprehensible instructions*: Describe the user's tasks in familiar terminology. Try to be brief; but if more information is needed, make a set of help screens available to the novice user. In support of brevity, just decribe the necessary action ("Type the address" or simply "address:") and avoid pronouns ("You should type the address") or references to the user "The user of the form should type the address." Another useful rule is to use the word *type* for entering information and *press* for special keys such as the TAB, ENTER, cursor movement, or Programmed Function (PFK, PF, or F) keys. Since ENTER often refers to the special key, avoid using it in the instructions (for example,

SKYHAWKE Ex. 1015, page 89

do not use "Enter the address," but stick to "Type the address.") Once a grammatic style for instructions is developed, be careful to apply that style consistently.

- *Logical grouping and sequencing of fields*: Related fields should be adjacent and aligned with blank space for separation between groups. The sequencing should reflect common patterns; for example, city followed by state followed by zip code.

- *Visually appealing layout of the form*: A uniform distribution of fields is preferable to crowding one part of the screen and leaving other parts blank. Alignment creates a feeling of order and comprehensibility. For example, the field labels, Name, Address, and City, were right justified so that the data entry fields would be vertically aligned. This allows the frequent user to concentrate on the entry fields and ignore the labels. If working from hard copy, the screen should match the paper form.

- *Familiar field labels*: Common terms should be used. If `Address` were replaced by `Domicile`, many users would be uncertain or anxious about what to do.

- *Consistent terminology and abbreviations*: Prepare a list of terms and acceptable abbreviations and diligently use the list, making additions only after careful consideration. Instead of varying such terms as `Address`, `Employee Address`, `ADDR.`, and `Addr.`, stick to one term, such as `Address`.

- *Visible space and boundaries for data entry fields*: Underscores or other markers indicate the number of characters available, so users will know when abbreviations or other trimming strategies are needed.

- *Convenient cursor movement*: A simple and visible mechanism is needed for moving the cursor, such as a TAB key or cursor movement arrows.

- *Error correction for individual characters and entire fields*: A backspace key and overtyping should be allowed to enable easy repairs or changes to entire fields.

- *Error messages for unacceptable values*: If users enter an unacceptable value, the error message should appear on completion of the field. The message should indicate permissible values of the field, for example, if the zip code is entered as 28K21 or 2380, the message might indicate that "Zip codes should have 5 digits."

- *Optional fields should be marked*: The word optional or other indicators should be visible. Optional fields should follow required fields, whenever possible.

- *Explanatory messages for fields*: If possible, explanatory information about a field or its values should appear in a standard position, such as in a window on the bottom, whenever the cursor is in the field.

- *Completion signal*: It should be clear to the users what to do when they are finished filling in the fields. Generally, designers should avoid automatic completion when the last field is filled, because users may wish to go back and review or alter field entries.

These considerations may seem obvious, but often forms designers omit the title or have unnecessary computer file names, strange codes, unintelligible instructions, unintuitive groupings of fields, cluttered layouts, obscure field labels, inconsistent abbreviations or field formats, awkward cursor movement, confusing error correction procedures, hostile error messages, and no obvious way to signal completion.

Detailed design rules should reflect local terminology and abbreviations; field sequences familiar to the users; the width and height of the display device; highlighting features such as reverse video, underscoring, intensity levels, color, and fonts; the cursor movement keys; and coding of fields.

3.9.2 Coded fields

Columns of information require special treatment for data entry and for display. Alphabetic fields are customarily left justified on entry and on display. Numeric fields may be left justified on entry but then become right justified on display. When possible, avoid entry and display of leftmost zeroes in numeric fields. Numeric fields with decimal points should line up on the decimal points.

Special attention should be paid to such common fields as:

- *Telephone numbers*: Offer a form to indicate the subfields:

```
Phone: (_ _ _) _ _ _-_ _ _ _
```

Be alert to such special cases as addition of extensions or the need for nonstandard formats for foreign numbers.

- *Social Security numbers*: The pattern for Social Security numbers should appear on the screen as:

```
Social Security Number: _ _ _-_ _-_ _ _ _
```

When the user has typed the first three digits, the cursor should jump to the leftmost position of the two-digit field.

- *Times*: Even though the twenty-four hour clock is convenient, many people find it confusing and prefer A.M. or P.M. designations. The form might appear as:

```
_ _:_ _  _ _ (9:45 AM or PM)
```

  Seconds may or may not be included, adding to the variety of necessary formats.

- *Dates*: This is one of the nastiest problems for which no good solution exists. Different formats for dates are

SKYHAWKE Ex. 1015, page 92

appropriate for different tasks, and European rules differ from American rules. It may take a generation until an acceptable standard emerges.

When presenting coded fields, the instructions might show an example of correct entry, for example:

```
Date: _ _/_ _/_ _  (04/22/86 indicates April 22, 1986)
```

For many people, examples are more comprehensible than an abstract description, such as MM/DD/YY.

- *Dollar amounts (or other currency)*: The dollar sign should appear on the screen, and users then type only the amount. If a large volume of whole dollar amounts are to be entered, the user might be presented with a field such as:

```
Deposit amount: $_ _ _ _ _.00
```

with the cursor to the left of the decimal point. As the user types numbers, they shift left. To enter an occasional cents amount, the user must type the decimal point to reach the 00 field for overtyping.

Other considerations in form fill-in design include dealing with multiscreen forms, mixing menus with forms, the role of graphics, relationship to paper forms, use of pointing devices, use of color, handling of special cases, and integration of a word processor to allow remarks.

3.10 PRACTITIONER'S SUMMARY

Begin by understanding the semantic structure of your application within the vast range of menu selection situations. Concentrate on organizing the sequence of menus to match the user's tasks, ensure that

SKYHAWKE Ex. 1015, page 93

each menu is a meaningful semantic unit, and create items that are distinctive and comprehensible. If some users make frequent use of the system, then typeahead, shortcut, or macro strategies should be allowed. Permit simple traversals to the previously displayed menu and to the main menu. Finally, be sure to conduct human factors tests and involve human factors specialists in the design process (Savage et al., 1982). When the system is implemented, collect usage data, error statistics, and subjective reactions to guide refinement.

Whenever possible, use a menu builder/driver system to produce and display the menus. Commercial menu creation systems are available and should be used to reduce implementation time, ensure consistent layout and instructions, and simplify maintenance.

3.11 RESEARCHER'S AGENDA

Experimental research could help refine the design guidelines concerning semantic organization and sequencing in single and linear sequences of menus. How can differing communities of users be satisfied with a common semantic organization when their information needs are very different? Should users be allowed to tailor the structure of the menus or is the advantage greater in compelling everyone to use the same structure and terminology?

Should a tree structure be preserved even if some redundancy is introduced? How can networks be made safe?

Research opportunities abound. Depth versus breadth tradeoffs under differing conditions need to be studied to provide guidance for designers. Layout strategies, wording of instructions, phrasing of menu items, use of color, response time, and display rate are all excellent candidates for experimentation. Exciting possibilities are becoming available with larger screens, multiple displays, and novel selection devices.

Implementers would benefit from the development of software tools to support menu system creation, management, usage statistics gathering,

and evolutionary refinement. Portability of "menu-ware" could be enhanced to facilitate transfer across systems.

REFERENCES

Billingsley, P. A., Navigation through hierarchical menu structures: Does it help to have a map? *Proc. Human Factors Society, 26th Annual Meeting*, (1982), 103–107.

Brown, C. Marlin, *Human-Computer Interface Design Guidelines*, Ablex Publishing Company, Norwood, NJ, (1986).

Brown, James W., Controlling the complexity of menu networks, *Communications of the ACM 25*, 7, (July 1982), 412–418

Card, Stuart K., User perceptual mechanisms in the search of computer command menus, *Proc. Human Factors in Computer Systems*, (March 1982), 190–196.

Clauer, Calvin Kingsley, An experimental evaluation of hierarchical decision-making for information retrieval, IBM Research Report RJ 1093, (September 15, 1972), 83 pages.

Doughty, Roger K., and Kelso, John, An evaluation of menu width and depth on user performance, Unpublished project paper done with Prof. James Foley, George Washington University, Washington, DC (1984).

Dray, S. M., Ogden, W. G., and Vestewig, R. E., Measuring performance with a menu-selection human-computer interface, *Proc. Human Factors Society, 25th Annual Meeting*, (1981), 746–748.

Galitz, Wilbert O., *Human Factors in Office Automation*, Life Office Managment Assn., Atlanta, GA, (1980).

Herot, Christopher F., Graphical user interfaces, in Vassiliou, Y. (Editor), *Human Factors and Interactive Computer Systems*, Ablex Publishers, Norwood, NJ, (1984), 83–103.

Hiltz, Starr Roxanne, and Turoff, Murray, The evolution of user behavior in a computerized conferencing system, *Communications of the ACM 24*, 11, (November 1981), 739–751.

Kiger, John I., The depth/breadth trade-off in the design of menu-driven user interfaces, *International Journal of Man-Machine Studies 20*, (1984), 201–213.

Koved, Lawrence, and Shneiderman, Ben, Embedded menus: Menu selection in context, *Communications of the ACM 29*, (1986) , 312–318.

Landauer, T. K., and Nachbar, D. W., Selection from alphabetic and numeric menu trees using a touch screen: Breadth, depth, and width, *Proc. Human Factors in Computing Systems* (April 1985), ACM SIGCHI, New York, 73–78.

Laverson, Alan, Norman, Kent, and Shneiderman, Ben, An evaluation of jump-ahead techniques for frequent menu users, University of Maryland Computer Science Technical Report 1591, (December 1985).

Lee, E., and Latremouille, S., Evaluation of tree structured organization of information on Telidon, *Telidon Behavioral Research I*, Department of Communications, Ottawa, Canada, (1980).

Liebelt, Linda S., McDonald, James E., Stone, Jim D., and Karat, John, The effect of organization on learning menu access, *Proc. Human Factors Society, 26th Annual Meeting*, (1982), 546–550.

McDonald, James E., Stone, Jim D., and Liebelt, Linda S., Searching for items in menus: The effects of organization and type of target, *Proc. Human Factors Society, 27th Annual Meeting*, (1983), 834–837.

McEwen, S. A., An investigation of user search performance on a Telidon information retrieval system, *Telidon Behavioral Research 2*, Ottawa, Canada, (May 1981).

Mantei, Marilyn, Disorientation behavior in person-computer interaction, Ph. D. Dissertation, University of Southern California (August 1982).

Martin, James, *Viewdata and the Information Society*, Prentice-Hall, Inc., Englewood Cliffs, NJ, (1982), 293 pages.

Miller, Dwight, P., The depth/breadth tradeoff in hierarchical computer menus, *Proc. Human Factors Society, 25th Annual Meeting*, (1981), 296–300.

Murray, Robert P., and Abrahamson, David S., The effect of system response delay and delay variability on inexperienced videotex users, *Behaviour and Information Technology 2*, 3, (1983), 237–251.

Ogden, William C., and Boyle, James M., Evaluating human-computer dialog styles: Command vs. form/fill-in for report modification, *Proc. Human Factors Society, 26th Annual Meeting*, Human Factors Society, Santa Monica, CA, (1982), 542–545.

Pakin, Sherwin E., and Wray, Paul, Designing screens for people to use easily, *Data Management*, (July 1982), 36–41.

Parton, Diana, Huffman, Keith, Pridgen, Patty, Norman, Kent, and Shneiderman, Ben, Learning a menu selection tree: Training methods compared, *Behaviour and Information Technology 4*, 2, (1985), 81–91.

Perlman, Gary, Making the right choices with menus, *INTERACT '84*, First IFIP International Conference on Human-Computer Interaction, North-Holland, Amsterdam (September 1984), 291–295.

Price, Lynne A., Design of command menus for CAD systems, *Proc. ACM-IEEE 19th Design Automation Conference*, (June 1982), 453–459.

Robertson, G., McCracken, D., and Newell, A., The ZOG approach to man-machine communication, *International Journal of Man-Machine Studies 14*, (1981), 461–488.

Savage, Ricky E., Habinek, James K., and Barnhart, Thomas W., The design, simulation, and evaluation of a menu driven user interface, *Proc. Human Factors in Computer Systems*, (1982), 36–40.

Shneiderman, Ben, Direct manipulation: A step beyond programming languages, *IEEE Computer 16*, 8, (August 1983).

Somberg, Benjamin, and Picardi, Maria C., Locus of information familiarity effect in the search of computer menus, *Proc. Human Factors Society, 27th Annual Meeting*, (1983), 826–830.

Teitelbaum, Richard C., and Granda, Richard, The effects of positional constancy on searching menus for information, *Proc. CHI '83, Human Factors in Computing Systems*, Available from ACM, Baltimore, MD (1983), 150–153.

Teitelbaum, T., and Reps, T., The Cornell program synthesizer: A syntax-directed programming environment, *Communications of the ACM 24*, 9, (September 1981), 563–573.

Tombaugh, Jo W., and McEwen, Scott A., Comparison of two information retrieval methods on Videotex: Tree-structure versus

SKYHAWKE Ex. 1015, page 97

alphabetic directory, *Proc. Human Factors in Computer Systems*, (1982), 106–110.

Young, R. M., and Hull, A., Cognitive aspects of the selection of Viewdata options by casual users, *Pathways to the Information Society, Proc. 6th International Conference on Computer Communication*, London, (September 1982), 571–576.

CHAPTER 4

COMMAND LANGUAGES

I soon felt that the forms of ordinary language were far too diffuse....I was not long in deciding that the most favorable path to pursue was to have recourse to the language of signs. It then became necessary to contrive a notation which ought, if possible, to be at once simple and expressive, easily understood at the commencement, and capable of being readily retained in the memory.

Charles Babbage, "On a method of expressing by signs the action of machinery," 1826

SKYHAWKE Ex. 1015, page 99

4.1 INTRODUCTION

The history of written language is rich and varied. Early tally marks and pictographs on cave walls existed for millenia before precise notations for numbers or other concepts appeared. The Egyptian hieroglyphs of 5,000 years ago were a tremendous advance because standard notations facilitated communication across space and time. Eventually, languages with a small alphabet and rules of word and sentence formation dominated because of the relative ease of learning, writing, and reading. In addition to these natural languages, special languages for mathematics, music, and chemistry emerged because they facilitated communication and problem solving. In the twentieth century, novel notations were created for such diverse domains as dance, knitting, higher forms of mathematics, logic, and DNA molecules.

The basic goals of language design are:

- precision
- compactness
- ease in writing and reading
- speed in learning
- simplicity to reduce errors
- ease of retention over time.

Higher level goals include:

- a close correspondence between reality and the notation
- convenience in carrying out manipulations relevant to the users' tasks
- compatibility with existing notations
- flexibility to accommodate novice and expert users
- expressiveness to encourage creativity
- visual appeal.

Constraints on a language include:

SKYHAWKE Ex. 1015, page 100

- the capacity for human beings to record the notation
- a good match with the recording and the display media (e.g., clay tablets, paper, printing presses)
- the convenience in speaking (vocalizing).

Successful languages evolve to serve the goals within the constraints.

The printing press was a remarkable stimulus to language development because it made widespread dissemination of written work possible. The computer is another remarkable stimulus to language development, not only because widespread dissemination through networks is possible, but also because the computer is a tool to manipulate languages and because languages are a tool for manipulating computers.

The computer has had only a modest influence on spoken natural languages, compared to its enormous impact as a stimulus to the development of numerous new formal written languages. Early computers were meant to do mathematical computations, so the first programming languages had a strong mathematical flavor. But computers were quickly found to be effective manipulators of logical expressions, business data, graphics, and text. Increasingly, computers are used to operate on the real world: directing robots, issuing dollar bills at bank terminals, controlling manufacturing, and guiding spacecraft. These newer applications encourage language designers to find convenient notations to direct the computer while preserving the needs of people to use the language for communication and problem solving.

Therefore, effective computer languages must not only represent the users' tasks and satisfy the human needs for communication but must also be in harmony with mechanisms for recording, manipulating, and displaying these languages in a computer.

Computer programming languages that were developed in the 1960s and early 1970s, such as FORTRAN, COBOL, ALGOL, PL/I, or Pascal, were designed for use in a noninteractive computer environment. Programmers would compose hundreds or thousands of lines of code, carefully check them over, and then compile or interpret by computer to produce a desired result. Incremental programming was one of the design considerations in BASIC and such advanced languages as LISP,

APL, or PROLOG. Programmers in these languages were expected to build smaller pieces online and interactively execute and test them. Still the common goal was to create a large program that was preserved, studied, extended, and modified.

Database query languages developed in the mid to late 1970s, such as SQL or QUEL, emphasized shorter segments of code (three to twenty lines) that could be written at a terminal and immediately executed. The goal of the user was more to create a result than a program.

Command languages, which originated with operating systems commands, are distinguished by their immediacy and by their impact on devices or information. Users issue a command and watch what happens. If the result is correct, the next command is issued; if not, some other strategy is adopted. The commands are brief and their existence is transitory. Of course, command histories are sometimes kept and macros are created in some command languages, but the essence of command languages is their ephemeral nature and the fact that they produce an immediate result on some object of interest.

Command languages are distinguished from menu selection systems by the fact that the users of command languages must recall notation and initiate action. Menu selection users receive instructions and must only recognize and choose among a limited set of visible alternatives; they respond more than initiate. Command language users are often called on to accomplish remarkable feats of memorization and typing. For example, does it make sense to type the UNIX command:

```
GREP -V ^$ FILEA > FILEB
```

in order to delete blank lines from a file? Similarly, to get printout on unlined paper with the IBM 3800 laser printer, a user at one installation was instructed to type

```
CP TAG DEV E VTSO LOCAL 2 OPTCD=J F=3871 X=GB12
```

The puzzled user was greeted with a shrug of the shoulders and the equally cryptic comment that "sometimes logic doesn't come into play, it's just getting the job done." This style of work may have been acceptable in the past, but user communities and their expectations are

SKYHAWKE Ex. 1015, page 102

changing. The empirical studies described in this chapter are beginning to clarify guidelines for many command language design issues.

Command languages may consist of single commands or have complex syntax (Section 4.2). The language may have only a few operations or thousands. Commands may have a hierarchical structure or permit concatentation to form variations (Section 4.3). A typical form is a verb followed by a noun object with qualifiers or arguments for the verb or noun. Abbreviations may be permitted (Section 4.5). Feedback may be generated for acceptable commands and error messages (see Section 8.1) may result from unacceptable forms or typos. Command language systems may offer the user brief prompts or they may be close to menu selection systems (Section 4.6). Finally, natural language interaction can be considered as a complex form of command language (Section 4.7).

4.2 FUNCTIONALITY TO SUPPORT USERS' TASKS

People use computers and command language systems to accomplish tasks. The most common application of command languages is for text editing; other applications include operating systems control, bibliographic retrieval, database manipulation, electronic mail, financial management, airline or hotel reservations, inventory, manufacturing process control, and adventure games.

The critical determinant of success is the functionality of the system. People will use a computer system if it gives them powers not otherwise available. If the power is attractive enough, people will use a system despite its poor user interface. Therefore, the first step for the designer is to determine the functionality of the system by assessing the user task domain.

A common design error is excess functionality. In a misguided effort to add features, options, and commands, the designer can overwhelm the user. Excess functionality means more code to maintain, potentially more bugs, possibly slower execution, and more help screens, error messages, and user manuals (see Chapter 9). For the user, excess functionality slows learning, increases the chance of error, and adds the confusion of longer manuals, more help screens, and less specific error

messages. On the other hand, insufficient functionality leaves the user frustrated because an apparent function is not supported. For instance, the system might require the user to copy the contents of the screen by hand because there is no simple print command or to reorder the output because there is no sort command.

Evidence of excessive functionality comes from a study of 17 secretaries at a scientific research center who used IBM's XEDIT editor for a median of 18 months for 50 to 360 minutes per day (Rosson, 1983). Their usage of XEDIT commands was monitored for five days. The average number of commands used was 26 per user with a maximum of 34; the number of commands was correlated with experience (r=.49). XEDIT has 141 commands, so even the most experienced user dealt with less than a quarter of the commands. Users did not appear to employ idiosyncratic subsets of the language but added commands to their repertoire in an orderly and similar pattern.

Careful task analysis might result in a table of user communities and tasks with each entry indicating expected frequency of use. The high volume tasks should be made easy to carry out, and then the designer must decide which communities of users are the prime audience for the system. Users may differ in their position in an organization, their knowledge of computers, or their frequency of system usage. One difficulty in carrying out such a task analysis is predicting who the users might be and what tasks they need to accomplish.

Inventing and supplying new functions are the major goals of many designers. They know that marketplace acceptance is often determined by the availability of functions that the competition does not provide. Word processor designers continue to add such functions as boldface, footnotes, dual windows, mail merge, or spelling checks to entice customers. A feature analysis list (Figure 4.1) can be very helpful in comparing designs and in discovering novel functions (Roberts, 1980).

At an early stage, the destructive operations such as deleting objects or changing formats should be carefully evaluated to ensure that they are reversible or at least protected from accidental invocation. Designers should also identify error conditions and prepare error messages. A transition diagram showing how each command takes the user to another state is a highly beneficial aid to design, as well as to eventual training of

TEXT EDITOR FEATURE LIST

Estimated time to install (15 minutes to two hours)
Number of diskettes provided (1 to 7)
Right to make copies
On-screen tutorial
Textbook tutorial
Textbook reference guide
Online help
Meaningful error messages
Spelling checker built in to word processor
Thesaurus built in to word processor
Mail merge
Automatic table of contents generation
Automatic index generation
Menu, command, or function key driven
Save block
Block defined by highlight or markers
Maximum size for block operation
Document size limit
Capacity to edit more than one file
Can rename disk files
Can copy disk files
Can show disk directory
What you see is what you get
Preview print format

Editing allowed during printing
Print part of file
Can chain documents for printing
Can queue documents for printing
Automatic page numbering
Print multiple copies
Automatic file save
Save file without exiting
Automatic backup file
Can create file without embedded codes
Subscript/superscript
Italics
Underscoring
Boldface
Multiple fonts
Multiple font sizes
Left and right justification
Centering
Tabbing
Proportional spacing
Multiple column output
Footnotes
Endnotes
Line spacing options
Number of printers supported
Characters per line range (78–455)
Lines per screen
Can change screen colors
Can redefine key functions
Can specify macros

Automatic hyphenization
Can switch from insert to overwrite modes
Automatic indentation
Multiple indents/outdents
Change case command
Ruler line can be displayed to show tabs
Display column, line, and page number
Headers and footers
Math functions
Sort functions
Move cursor by character, word, sentence, paragraph
Move cursor by screen
Move cursor to left or right ends of line
Move cursor to top or bottom of screen
Move cursor to top or bottom of document
Delete by character, word, line, sentence, or paragraph
Delete to end of document
Undelete
Search forward and backward
Search by patterns
Can ignore case in searching
Leave and locate markers
Copy/move
Copy/move by columns

Figure 4.1: Feature list for word processors distilled from Roberts (1980) and Wiswell, Phil, Word processing: The latest word, *PC Magazine*, (August 20, 1985), 110–134.

SKYHAWKE Ex. 1015, page 105

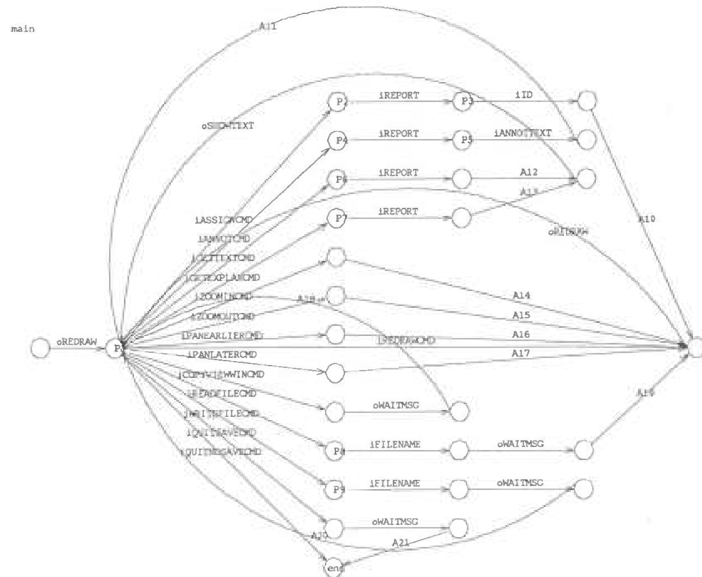


Figure 4.2: This transition diagram indicates user inputs with an "i" and computer outputs with an "o". This is a relatively simple diagram showing only a portion of the system. Complete transition diagrams may take many pages. (Courtesy of Robert J. K. Jacob, Naval Research Laboratory, Washington, DC)

users (Figure 4.2). If the diagram gets too complicated, it may signal the need to redesign the system.

Major considerations for expert users are the possibilities of tailoring the language to suit personal work styles and creating named macros to permit several operations to be carried out with a single command. Macro facilities allow extensions that the designers could not foresee or that are beneficial to only a small fragment of the user community. A macro facility can be a full programming language that might include specification of arguments, conditionals, iteration, integers, strings, and screen manipulation primitives, plus library and editing tools. Well-developed macro facilities are one of the strong attractions of command languages.

4.3 COMMAND ORGANIZATION STRATEGIES

Several strategies for command organization have emerged, but guidelines for choosing among these are only beginning to be discussed. A unifying concept, model, or metaphor is an aid to learning, problem solving, and retention (Carroll & Thomas, 1982). In electronic mail circles, lively discussions can be started over the metaphoric merits of such task-related objects as file drawers, folders, documents, memos, notes, letters, or messages. The debates continue about the appropriate task domain actions (CREATE, EDIT, COPY, MOVE, DELETE) and the choice of an action pair LOAD/SAVE (too much in the computer domain), READ/WRITE (acceptable for letters but awkward for file drawers), or OPEN/CLOSE (acceptable for folders but awkward for notes).

Similarly, debate continues over whether the commands should manipulate lines, as in program editors and older line-oriented editors, or words, sentences, and paragraphs, as in new word processors. Choosing one strategy over another is helpful. Designers who fail to choose and attempt to support every possibility risk overwhelming the users while missing the opportunity to optimize for one strategy. Designers often err by choosing a metaphor closer to the computer domain rather than the user's task domain. Of course, metaphors can mislead the user, but careful design can reap the benefits while reducing the detriments.

Having adopted a concept, model, or metaphor for operations, the designer must now choose a strategy for the command structure. Mixed strategies are possible, but learning, problem solving, and retention may be aided by limiting the complexity.

4.3.1 Simple command list

Each command is chosen to carry out a single task, and the number of commands matches the number of tasks. With a small number of tasks, this approach can produce a system that is simple to learn and use. With a large number of commands, there is danger of confusion. The VI editor on UNIX systems offers many commands while attempting to keep

SKYHAWKE Ex. 1015, page 107

VI COMMANDS TO MOVE THE CURSOR

Moving within a window

H	go to home position (upper left)
L	go to last line
M	go to middle line
(CR)	next line (carriage return)
+	next line
−	previous line
CTRL−P	previous line in same column
CTRL−N	next line in same column
(LF)	next line in same column (line feed)

Moving within a line

0	go to start of line
$	go to end of line
(space)	go right one space
CRTL−H	go left one space
h	go left one space
w	forward one word
b	backward one word
e	end (rightmost) character of a word
)	forward one sentence
(	backward one sentence
}	forward one paragraph
{	backward one paragraph
W	blank out a delimited word
B	backwards blank out a delimited word
E	go to the end of a delimited word

Finding a character

fx	find the character x going forward
Fx	find the character x going backward
tx	go up to x going forward
Tx	go up to x going backward

Scrolling the window

CTRL–F	go forward one screen
CTRL–B	go backward one screen
CTRL–D	go forward one half screen
CTRL–U	go backward one half screen
G	go to line
/pat	go to line with pattern forward
pat	go to line with pattern backward

Figure 4.3: The profusion of commands in vi may enable expert users to get tasks done with just a few actions, but the number of commands can be overwhelming to novice and intermittent users.

the number of keystrokes low. This results in complex strategies employing single letters, shifted single letters, and CTRL key plus single letters (Figure 4.3). Furthermore, some commands stand alone, whereas others must be combined in often irregular patterns.

4.3.2 Command plus arguments

Each command (COPY, DELETE, PRINT, . . .) is followed by one or more arguments (FILEA, FILEB, FILEC, . . .) that indicate objects to be manipulated.

```
COPY FILEA,FILEB
DELETE FILEA
PRINT FILEA,FILEB,FILEC
```

Commands may be separated from the arguments by a blank or other delimiter, and the arguments may have blanks or delimiters between them (Schneider et al., 1984). Keyword labels for arguments may be helpful to some users; for example;

```
COPY FROM=FILEA TO=FILEB
```

The labels require extra typing and increase chances of a typo, but readability is improved and order dependence is eliminated.

4.3.3 Command plus options and arguments

Commands may have options (3, HQ, . . .) to indicate special cases. For example

```
PRINT/3,HQ FILEA
PRINT (3,HQ) FILEA
PRINT FILEA -3,HQ
```

may produce three copies of FILEA at the printer in the headquarters building. As the number of options grows, the complexity can become overwhelming and the error messages less specific. The arguments may also have options, such as version numbers, privacy keys, or disk addresses.

The number of arguments, number of options, and the permissible syntactic forms can grow very rapidly. One airline reservations system uses the following command to locate availability of flight on August 21, from Washington's National Airport (DCA) to New York's La Guardia Airport (LGA) around 3:00 P.M.:

```
A0821DCALGA0300P
```

Even with substantial training, error rates can be high with this approach, but frequent users seem to manage and even appreciate the compact form of this type of command.

UNIX is a widely used command language system in spite of the complexity of the command formats (see Figure 4.4) that have been severely criticized (Norman, 1981). Here again, users will master complexity to benefit from the rich functionality in a system. Error rates with actual use of UNIX commands ranged from 3 percent to 53 percent (Kraut et al., 1983; Hanson et al., 1984). Even common commands generated high syntactic error rates: mv (18 percent), cp (30 percent), and awk (34 percent). Still, there is a certain attraction to the complexity

SKYHAWKE Ex. 1015, page 110

at 2A Friday timehog
at 2:00 a.m. Friday, run program
awk '{print $1 + $2}' file1
print sum of first two fields of each line
cat −n file1
print specified file to terminal, number output lines
cat file1 >> file2
append **file1** *to end of* **file2**
cc file.c
compile C program, executable in **a.out**
cd /usr/lib
change working directory to specified one
chmod g+rw file1 file2
change mode of files, adding group read and write access
chmod 600 file
change mode of file, allow only read and write by owner
cp file1 file2
make a copy of **file1** *named* **file2**
cp −r dir /tmp
recursively copy **dir** *and its subdirectories to* **/tmp**
diff −l dir1 dir2
summarize differences between files in **dir1** *and* **dir2**
f77 file.f
compile Fortran program, executable in **a.out**
f77 −o file file.f file.o
compile Fortran program, link with **file.o**, *executable in* **file**
find $HOME −name '#*' −exec rm { } \;
remove files with names beginning with a pound sign
finger name
look up information on user's login or real name
grep '[Pp]hone' file
print all lines in file containing **Phone** *or* **phone**
grep −l main *
print names of files in current directory containing **main**
head −6 file
print first six lines of file
kill −9 0
send a KILL signal to processes started since login
ln −s file1 file2 /tmp
make symbolic links to files in specified directory
lpq job user
report print spooler status of user's job
lpr −p file
paginate file and spool it to the line printer
ls
print a list of the files in the current directory
ls −R /bin
list files in specified directory and its subdirectories
mail molly tracey < file
send a file to specified users as mail
man spell
print Unix user's manual page for a command
mkdir /tmp/myjunk
make a new directory
more +50 file
view file by screenful, starting at line 50
mv file1 file2 /tmp
move files to specified directory
nroff file ¦ more
preview formatted file on terminal
pc file.p
compile Pascal program, executable in **a.out**
pr file ¦ lpr
paginate a file with default header, spool output
ps l
print long listing of current processes, PID's and status
pwd
print current working directory
rlogin puter2
login on remote computer
rm file
remove (delete) a file
rm −i junk[0−9]
remove files **junk0** ... **junk9**, *confirming first*
sort +3 −4 file
print file sorted only on fourth field
stty everything
print all stty option settings
stty raw; prog; stty −raw
set terminal to **raw** *mode, run a program, and restore mode*
style −p file
print sentences in file containing a passive verb
vi file
edit file using full screen editor
w
list who's logged in, and what they're doing

Figure 4.4: Examples of common UNIX commands with brief explanations. (Courtesy of Specialized Systems Consultants, Inc., Seattle WA)

among a portion of the potential user community. There is satisfaction in overcoming the difficulties and becoming one of the inner circle ("gurus" and "wizards") who are knowledgeable about system features—command language macho.

4.3.4 Hierarchical command structure

The full set of commands is organized into a tree structure, like a menu tree. The first level might be the command action, the second

might be an object argument, and the third might be a destination argument:

Action	Object	Destination
CREATE	File	File
DISPLAY	Process	Local printer
REMOVE	Directory	Screen
COPY		Laser printer
MOVE		

If a hierarchical structure can be found for a set of tasks, it offers a meaningful structure to a large number of commands. In this case, 5 x 3 x 4 = 60 tasks can be carried out with only five command names and one rule of formation. Another advantage is that a command menu approach can be developed to aid the novice or intermittent user, as was done in VisiCalc and Lotus 1–2–3.

Several help systems allow a hierarchical command to retrieve text about subsystems and their commands. For example, to get help on the editor command for deleting lines in a document, the user might type:

```
HELP EDIT DELETE LINES
```

Of course, the difficulty comes in knowing what keywords are available. Users can type the first few elements of the command and then receive a menu of items.

Many word processors use a hierarchical command structure for the numerous commands that they support. For example, Figure 4.5 shows the command structure for FinalWord.

4.4 THE BENEFITS OF STRUCTURE

Human learning, problem solving, and memory are greatly facilitated by meaningful structure. If command languages are well-designed, users

SKYHAWKE Ex. 1015, page 112

```
Menus - CTRL-X              Buffers Menu
  Buffers                     Delete buffer            Switch buffer
  Capitalization              List buffers             Z - exit menu
  Files                       Previous buffer
  Help
  Layout                    Capitalization Menu
  Miscellaneous               Clear highlighting       Set highlighting
  Regions                     Flip highlighting        Uppercase
  Set                         Lowercase                Z - exit menu
  Windows
  Z - exit menu             Files Menu
                              Backup and save file     Save file
                              Directory edit           Write file
                              Find file                Z - exit menu
                              Read file

                            Help Menu
                              Explain key              Remove help buffer
                              Help about               Z - exit menu

                            Layout Menu
                              Advanced print           Print
                              Center line              Right flush line
                              Fill paragraph           Unjustify paragraph
                              Justify paragraph        Verify advanced print
                              Left flush line          Z - exit menu

                            Miscellaneous Menu
                              Abort printing           Line count
                              Center point             Query replace
                              Display refresh          Report position
                              Exit editor              Z - exit menu
                              Global replace

                            Regions Menu
                              Append to KILLS          Outdent
                              Copy to marker           Set marker
                              Delete to marker         Undelete
                              Indent

                            Set Menu
                              Fill mode on/off         Overwrite mode on/off
                              Highlight on/off         Report values
                              Indent column            Tab interval
                              Line length              Z - exit menu

                            Windows Menu
                              Grow window              Switch windows
                              Move other               Two windows
                              One window               Z - exit menu
```

Figure 4.5: The tree structure of menus in the FinalWord word processor (Mark of the Unicorn). For example, by typing CTRL-X F W, the user can write the file out to the disk. The menus are shown if the user pauses for more than three seconds between the three keypresses.

can recognize the structure and easily encode it in their semantic knowledge storage. For example, if users can uniformly edit such objects as characters, words, sentences, paragraphs, chapters, and documents, this meaningful pattern is easy to learn, apply, and recall. On the other hand, if they must overtype a character, change a word, revise a

sentence, replace a paragraph, substitute a chapter, and alter a document, then the challenge grows substantially, no matter how elegant the syntax (Scapin, 1982).

Meaningful structure is beneficial for task concepts, computer concepts, and syntactic details of command languages. Yet, many sytems fail to provide a meaningful structure. One widely used operating system displays various information as a result of forms of the `LIST`, `QUERY`, `HELP`, and `TYPE` commands, and moves objects as a result of the `PRINT`, `TYPE`, `SPOOL`, `PUNCH`, `SEND`, `COPY`, or `MOVE` commands. Defaults are inconsistent for different features, four different abbreviations for `PRINT` and `LINECOUNT` are required, binary choices vary between `YES/NO` and `ON/OFF`, and function key usage is inconsistent. These flaws emerge from multiple uncoordinated design groups and insufficient attention by the managers, especially as features are added over time.

An explicit list of design conventions can be an aid to designers and managers. Exceptions may be permitted, but only after thoughtful discussions. Users can learn systems with inconsistencies, but more slowly and with greater chance of making mistakes. One difficulty is that there may be conflicting design conventions.

4.4.1 Consistent argument ordering

Choices among conventions can sometimes be resolved by experimentation with alternatives (Barnard et al., 1981). A command language with six functions, each requiring two arguments, was developed for decoding messages. One argument was always a message identification number and the other argument was a file number, code number, digit, and so on. In normal English usage, the message identification would sometimes be the direct object of an explanatory sentence, such as SAVE the MESSAGE ID with this REFERENCE NUMBER. So, one rule of consistent command formation was to follow English usage. The second consistency rule was to have the message identification always as the first or always as the second argument. This resulted in four possible command groups:

SKYHAWKE Ex. 1015, page 114

```
DIRECT OBJECT ARGUMENT FIRST     DIRECT OBJECT ARGUMENT SECOND

SEARCH  file no,message id       SEARCH  message id,file no
TRIM    message id,segment size  TRIM    segment size,message id
REPLACE message id,code no       REPLACE code no,message id
INVERT  group size,message id    INVERT  message id,group size
DELETE  digit,message id         DELETE  message id,digit
SAVE    message id,reference no  SAVE    reference no,message id

CONSISTENT ARGUMENT FIRST        CONSISTENT ARGUMENT SECOND

SEARCH  message id,file no       SEARCH  file no,message id
TRIM    message id,segment size  TRIM    segment size,message id
REPLACE message id,code no       REPLACE code no,message id
INVERT  message id,group size    INVERT  group size,message id
DELETE  message id,digit         DELETE  digit,message id
SAVE    message id,reference no  SAVE    reference no,message id
```

Forty-eight female subjects used one of these systems for an hour to decode messages. Actually, half the subjects had variant command names, such as SELECT instead of SEARCH, but this manipulation was a minor effect. Time to perform tasks decreased during the ten trials, but the speed-up was consistent across command styles. The results strongly favored using consistent argument positions rather than the consistent direct object position, suggesting that English language rules of formation were not as effective as the simpler positional rule. The shortest task times, fewest help requests, and fewest errors occurred with the consistent argument first. These results lead to the conjecture that command languages should allow users to express the simple, familiar, or well-understood features first, and then allow users to consider the more varying aspects.

Follow-up studies by the same group (Barnard et al., 1981; Barnard et al., 1982) replicated the results about positional consistency and pursued several related issues. One frequent design consideration is whether the command verb or the object of interest should come first. Command first form would be `DISPLAY FILE` or `INSERT LIST`; the object first form would be `FILE DISPLAY` or `LIST INSERT`. The evidence supports the command first strategy used in most languages and the principle that there is a fixed order. Allowing users the freedom to put

the command and object in either order generated more requests for help than fixing the order. Subjects pressed PF keys to initiate commands and select objects, so a further replication is necessary to validate the result if they had to remember and type commands. Mitigating factors may be the relative number of commands and objects and the familiarity the user has with each.

Finally, pilot studies by Jim Foley at George Washington University suggest that object first may be more appropriate when using selection by pointing on graphic displays. Different thinking patterns may be engaged in using visually oriented (right brain) interfaces than in using 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 fifteen 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.

SKYHAWKE Ex. 1015, page 116

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

Table 4.1: Impact of revised text editor commands on three levels of users (Ledgard et al., 1980)

The results (Table 4.1) clearly favored the keyword editor, indicating that command formation rules do make a difference. Unfortunately, no specific guidelines emerge except to avoid using unfamiliar symbols for new users of text editors, even if they 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 effected by syntactic variations.

4.4.3 Hierarchicalness and congruence

Carroll (1982) altered two design variables to produce four versions of a sixteen-command language for controlling a robot (Table 4.2). Commands could be hierarchical (verb-object-qualifer) or nonhierarchical (verb only) and congruent (for example, ADVANCE/RETREAT or RIGHT/LEFT) or non-congruent (GO/BACK or TURN/LEFT). Carroll uses congruent to refer to meaningful paris of opposites. Hierarchical structure and congruence (symmetry might be a better term) 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.

CONGRUENT		NONCONGRUENT	
Hierachical	Nonhierarchical	Hierachical	Nonhierarchical
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

Table 4.2: Command sets and partial results from Carroll (1982).

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

SKYHAWKE Ex. 1015, page 118

This study assessed performance of new users of a small command language. Congruence helped subjects 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 will be facilitated by the hierarchical structure and congruence.

Carroll's study was conducted during a half-day period; with a week of regular use, it is probable that differences would be substantially reduced. However, with intermittent use or 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 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 first-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 & Payne, 1984). Language L4 (Figure 4.6) is a subset of the commercial word processor based on EMACS, but it uses several conflicting organizing principles. Language L3 is simplified by using only the CTRL key, and it uses congruence and mnemonic naming where possible. Language L2 uses CTRL to mean forward and META for backwards, but mnemonicity is sacrificed. Language L1 uses the same meaningful structure for CTRL and META, congruent pairs, and mnemonicity.

Forty undergraduate subjects with no word processing experience were given twelve 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

SKYHAWKE Ex. 1015, page 119

	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. (Green and Payne, "Organization and learnability in computer languages," *International Journal of Man-Machine Studies [1984] 21*, 7–18. Used by permission of Academic Press Inc. [London] Limited.)

commands as possible. This was followed by presentation of the command descriptions with the request to write down the associated command syntax. The free recall and prompted recall tasks were both repeated. The results showed a statistically significant difference ($p < .001$) for languages, with L4 having the worst performance. The best performance was attained with L1 having the most structure. An online test would have been a useful follow-up to demonstrate the advantage in practice and over a longer period of time.

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

- 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. Up-arrow or down-arrow are highly suggestive of function, as are characters such as right- and left-angle-bracket, the plus sign, or 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, Michael L. 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

SKYHAWKE Ex. 1015, page 121

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 sometimes 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 in which variable name choices were less important in small modules with from ten to twenty 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: Old (delete, append, substitute), a new supposedly improved set (omit, add, change), and a random set designed to be confusing (allege, cipher, and 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 only to 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 twelve commands, subjects were more likely to recall and recognize the meaning of specific commands than general commands (Barnard et al., 1982).

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

Infrequent, discriminating	insert	delete
Frequent, discriminating	add	remove
Infrequent, nondiscriminating	amble	perceive
Frequent, nondiscriminating words	walk	view
General (frequent, nondiscriminating) words	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 performed worst 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. This means that commands should use brief and kinesthetically easy codes. Commands requiring shifted keys or CTRL keys, special characters, or difficult sequences are likely to have higher error rates. For text editing, when

SKYHAWKE Ex. 1015, page 123

many commands are applied and speed is appreciated, single-letter approaches are very attractive. Overall, brevity is a worthy goal since it can speed entry and possibly reduce error rates. Many word processor designers have pursued this approach even when mnemonicity was sacrificed, thereby making it difficult for novice and intermittent users.

In less demanding applications, designers have used longer command abbreviations, hoping that the gains in recognizability were 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 those who could create their own abbreviations (Grudin & 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 informs 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 and they stick with the full form of the command.

With experience and frequent use, abbreviations become attractive 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 & Porcu, 1982; Benbasat & Wand, 1984; Schneider, 1984). Potential strategies are:

1. Simple truncation: use the first, second, third, etc. letters of each command. This strategy requires that each command

SKYHAWKE Ex. 1015, page 124

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 ST for SORT.
4. First letter of each word in a phrase: this popular technique often fits with a hierarchical design plan.
5. Standard abbreviations from other contexts: 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 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 good as with vowel dropping (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 and a *simple* secondary rule used for those items where there is a conflict.
2. Abbreviations generated by the secondary rule should have a marker (e.g., an asterisk) incorporated in them.
3. The number of words abbreviated by the secondary rule should be kept to a minimum.

SKYHAWKE Ex. 1015, page 125

4. Users should be familiar with the rules used to generate abbreviations.
5. Truncation is an easy rule for users to work with, but it may also produce a large number of identical abbreviations for different words.
6. Fixed length abbreviations are preferable 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. Abbreviations are more likely to be used if users are confident in their knowledge of the abbreviations and if the benefit is more than a savings of one to two characters (Benbasat & 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 carefully evaluated 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 online version of the Official Airline Guide uses such prompts as:

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

This prompt is to remind users of the commands related to fares that have been displayed and the related flight schedules

```
+     move forward one screen
L#    limitations on airfares
```