Chapter 8

Command and Natural Languages

The Basic Goals of Language Design

- Precision
- Compactness
- Ease in writing and reading
- Speed in learning
- Simplicity to reduce errors
- Ease of retention over time

Computers and Languages

- After printing press, computer has been a remarkable stimulus to language development, because
  - widespread dissemination through networks is possible,
  - computers are a tool to manipulate languages, and
  - *languages are a tool to manipulate computers*
- Computers impact is mostly on the development of numerous formal written languages – from mathematical ones to those that help in operating real world activities.
  - High-level programming languages (procedural and OO)
  - Scripting languages
  - Database-query languages
  - Command languages

Command Language Examples

- Web addresses:
  - http://www.ccse.kfupm.edu.sa/~jauhar
- Unix commands
  - To delete blank lines from a file
    - **`grep –v ^$ filea > fileb`**
  - To print on a laser printer (in a particular installation)
    - **`CP TAG DEV E VTSO LOCAL 2 OPTCD=J F=3871 X=GB12`**

- The development of new command languages has slowed dramatically due to the emergence of direct-manipulation and menu-selection interfaces
- but still there are millions of users of command languages

Good Command Languages

- Command languages should be designed to suit the users' operation
- They should have some structure
  - hierarchical, concatenation
- Meaningful structure is highly beneficial
- Permitting abbreviations maybe useful
- Feedback should be generated for acceptable and unacceptable commands

Functionality to Support User's Tasks

- Users do wide range of work:
  - text editing, electronic mail, financial management, airline or hotel reservations, inventory, manufacturing process control, gaming, and so on.
- 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 a poor user interface
- Therefore, the first step for the designer is to determine the functionality of the system by studying the users' task domain.

Functionality to Support User's Tasks

- Providing excessive functionality is a common design error;
  - it slows learning,
  - increases the chances of error,
  - requires longer manuals, and more help screens
- On the other hand, insufficient functionality may leave the user frustrated because desired functionality may not be supported
- Careful task analysis might result in a table of user communities and tasks, with each entry indicating expected frequency
  - Make high-volume tasks easy to carry out
  - evaluate *destructive actions* to ensure reversibility, or at least are protected from accidental invocation

Command-Organization Strategies

- Simple command set
  - Each command is chosen to carry out a single task. The number of commands match the number of tasks.
  - Example:
    - In the vi editor of Unix
      - fx find the character x going forward
      - Fx find the character x going backward
  - For small number of tasks, this can produce a system easy to learn and use. However, large number of commands may result confusion.

Command-Organization Strategies

- Command plus arguments/options
  - Follow each command by one or more arguments that indicate objects to be manipulated, e.g.
    - **`COPY FILEA, FILEB`**
    - **`DELETE FILEA`**
  - Keyword labels for arguments are helpful for some users, e.g.
    - `COPY FROM=FILEA TO=FILEB`
  - Commands may also have options to indicate special cases, e.g.
    - **`PRINT/3,HQ FILEA`**
    - **`PRINT (3, HQ) FILEA`**
  - Error rates and the need for extensive training increase with the number of possible options
  - Frequent users appreciate compact commands
    - **`A0821DCALGA0300P`**
    - Checking seat availability on a flight on August 21, from Washington's National Airport (DCA) to New York's LaGuardia Airport (LGA) at about 3:00P.M.

Command-Organization Strategies

- Hierarchical command structure
  - The full set of commands is organized into a tree structure
    - First level: command action
    - Second level: object argument
    - Third level: destination argument
  - It offers a meaningful structure to a large number of commands
    - 5x3x4 = 60 tasks with 5 command names and 1 rule of formation

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

The Benefits of Structure: Consistent Argument Ordering

Human learning, problem solving, and memory are greatly facilitated by meaningful structure.

- Studies have shown that users perform significantly faster with **consistent argument ordering**

Inconsistent order of arguments	**Consistent order of arguments**
SEARCH file no, message id	SEARCH message id, file no
TRIM message id, segment size	TRIM message id, segment size
REPLACE message id, code no	REPLACE message id, code no
INVERT group size, message id	INVERT message id, group size

The Benefits of Structure: Symbols versus keywords

- Experiments show that performance improved in using keywords instead of symbols

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

	Percentage of Task Completed		Percentage of Erroneous Commands	
	Symbol	Keyword	Symbol	Keyword
Inexperienced users	28	42	19.0	11.0
Familiar users	43	62	18.0	6.4
Experienced users	74	84	9.9	5.6

The Benefits of Structure: Hierarchical structure and congruence

Carroll (1982)

CONGRUENT		NONCONGRUENT	
Hierarchical	Non-hierarchical	Hierarchical	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 Scores 14.88	14.63	7.25	11.00
Errors 0.50	2.13	4.25	1.63
Omissions 2.00	2.50	4.75	4.15

The Benefits of Structure: Hierarchical structure and congruence

- Congruence helped to remember the natural pairs of concepts and terms
- The hierarchical structure enabled subjects to master many commands with few keywords and one rule of formation.
- Retention should be facilitated by hierarchical structure and congruence

Sources of structure that have proved advantageous include:

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

Naming and Abbreviations

- There is often a lack of consistency or obvious strategy for construction of command abbreviations.
  - e.g., Unix commands: cp, ls, mkdir, cd, rm, pwd, …

Specificity Versus Generality:

Specific terms can be more descriptive, and if they are more distinctive, they maybe more memorable

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-ab

Abbreviation Strategies

- Command names should not only be meaningful for human learning and retention, they must also be in harmony with the mechanism for expressing them to the computer
  - commands should be easy to type fast and without errors
    - SHIFT or CTRL keys maybe difficult to type
  - novice users prefer full command names
  - short commands are appreciated by power users
  - sometimes, when both short & full command names are provided, users prefer to use the full ones

Abbreviation Strategies

- Six potential strategies for abbreviations:

1. **Simple truncation**: The first, second, third, etc. letters of each command.
2. **Vowel drop with simple truncation**: Eliminate vowels and use some of what remains.
3. **First and last letter**: Since the first and last letters are highly visible, use them.
4. **First letter of each word in a phrase**. Acronym technique
5. **Standard abbreviations from other contexts**: Use familiar abbreviations.
6. **Phonics**: Focus attention on the sound.

Guidelines for using abbreviations

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

- 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.
- Abbreviations generated by the secondary rule should have a marker (for example, an asterisk) incorporated in them.
- The number of words abbreviated by the secondary rule should be kept to a minimum.
- Users should be familiar with the rules used to generate abbreviations.
- Truncation should be used because it is an easy rule for users to comprehend and remember. However, when it produces a large number of identical abbreviations for different words, adjustments must be found.
- Fixed-length abbreviations should be used in preference to variable-length ones.
- Abbreviations should not be designed to incorporate endings (ING, ED, S).
- Unless there is a critical space problem, abbreviations should not be used in messages generated by the computer and read by the user.

Commands menus and keyboard shortcuts

- To relieve the burden of memorization of commands, some designers offer users brief prompts to available commands, in a format called *command menu*
  - Example:

    H)elp O)ptions P)rint G)o M)ain Screen Q)uit

  - Good for expert and intermittent users. Not much useful for novices.
- Keyboard shortcuts in most GUIs become a kind of menu for experienced users

Skipped section

- The following section has been skipped
  - 8.6 Natural Language in Computing