Lecture 7: Command Languages and Usability

Chapter 7

Command Language

- Definition
  - Interactive communication with a computer that requires the user to recall the notation and initiate the action by keyboarding textual elements. Command languages are typically interpreted a single action at a time.

Example Command Language (UNIX)

> ls -l *.*

foo.dat slides.prt exer.prt

> rm foo.dat

> ls -l *.*

slides.prt exer.prt

Elements of Language

- Lexicon
  - Words and punctuation
- Syntax
  - Sequence of words to create a correct sentence
- Semantics
  - "meaning" of a sentence based on the words
- Pragmatics
  - How sentences are used in sequence (discourse)
  - Context
  - Inference

Example Command Language (UNIX)

> ls -l *.*

foo.dat slides.prt exer.prt

> rm foo.dat

> ls -l *.*

slides.prt exer.prt

Lexical, syntax, semantics, pragmatics……

Usability Questions

- Does the language support necessary functions?
- Is it fast to enter a command?
- Is it easy to recognize what the command might do?
- Is it easy to recall a command?
- Are there few errors when using the language?

UNIX: A case study

- Study done at Bell Labs in 1981-1982
  - Automatic logging of all Unix command transactions in the lab
  - Analyzed
    - Frequency of command usage
    - Transitions between commands (tasks)
    - Error rates of commands
- Published by Kraut et al. In *Computer-Human Interaction Conference* (CHI) proceedings1983.

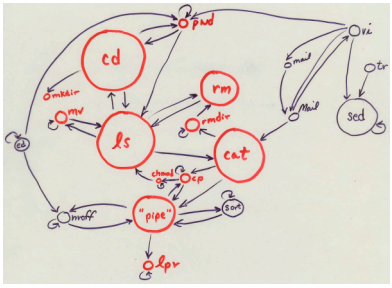
UNIX command usage

(Kraut et al. CHI '83)

Command	Frequency
cd	0.123
ls	0.100
cat	0.096
| "pipe"	0.062
vi	0.059
ed	0.056
rm	0.038
;	0.027
> "redirect"	0.025
Mail	0.020
nroff	0.015
mail	0.014
mv	0.012
grep	0.012
pwd	0.007
* "generic name"	> 0.005
cp	> 0.005
lpr	> 0.005
chmod	> 0.005
rmdir	> 0.005
mkdir	> 0.005

UNIX command transitions

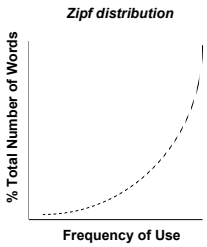
(Kraut et al. CHI '83)



UNIX

(Kraut et al. CHI '83)

- 400+ possible commands
- 20 commands (5%) account for 70% of usage
- 14 commands (3.5%) account for 50% usage



UNIX error rates

(Kraut et al. CHI '83)

- Types of errors
  - Lexical errors (error in entering command name, abbreviation, parameter specification). Gives message.
  - Syntactic errors (error in expression of a command such as missing parenthesis, wrong order of parameters) . Gives message.
  - Semantic errors (valid lexical and syntactic commands but errors where you don't get what you want). No message.
- Error rates for individual commands ranged from 3% to 50% for expert users!

Usability Problems with UNIX

- Lexicon: Abbreviation not suggestive of function
  - terse
  - inconsistent
  - jargon
- Syntax: Complex syntax
  - Action modifier(s) object(s)
- Semantics: Underutilization of commands
  - Unnecessary complexity to support many functions leads to complexity of most frequent
  - Hard to map commands to tasks
- Pragmatics: Lack of feedback

What this study doesn't tell us

- How hard it is to learn Unix
  - How much time does it take to get skilled?
  - Different types of users
- How to improve the design

The Basic Goals of Language Design (Chap. 7.1)

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

Higher-Level Goals of Language Design (Chap. 7.1)

- Close correspondence between reality and the notation
- Convenience in carrying out manipulations relevant to user's tasks
- Compatibility with existing notations -- "consistency"
- Flexibility to accommodate novice and expert users
- Expressiveness to encourage creativity
- Visual appeal

Aspects of Design

- Functionality (Semantics and Pragmatics)
  - Chap 7.2
- Syntax
  - Chap 7.2
- Lexicon
  - Chap 7.3

Functionality to Support User's Tasks (Chap 8.2)

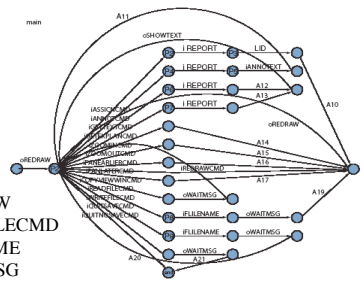
- determine functionality of the system by studying users' task domain
  - Text editing, operating system, gaming, airline reservation, database query
- create a list of task actions and objects
- abstract this list into a set of interface actions and objects
- represent low-level interface syntax
- create a table of user communities and tasks, with expected use frequency
- determine hierarchy of importance of user communities (i.e. prime users)
- evaluate destructive actions (e.g. deleting objects) to ensure reversibility
- identify error conditions and prepare error messages
- allow shortcuts for expert users, such as macros and customizing system parameters

Figure 8.1 Task Transition Diagram

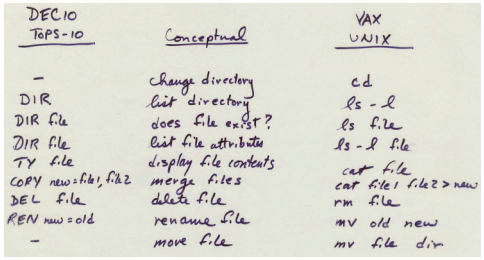
(I think the UNIX one in lecture 7 is better)

o is output
i is input

Example:
oREDRAW
iREADFILECMD
iFILENAME
oWAITMSG



Conceptual actions vs. commands



DEC10 TOPS-10	Conceptual	VAX UNIX
–	change directory	cd
DIR	list directory	ls -l
DIR file	does file exist?	ls file
DIR file	list file attributes	ls -l file
TY file	display file contents	cat file
COPY new=file1,file2	merge files	cat file1 file2 > new
DEL file	delete file	rm file
REN new=old	rename file	mv old new
–	move file	mv file dir

Syntax: Command-Organization Strategies (Chap 7.2)

A unifying interface concept or metaphor aids
- learning
- problem solving
- retention

Designers often err by choosing a metaphor closer to machine domain than to the user's task domain.

Good metaphor: Desktop with folders, files, trashcan

Types of command structure
- Simple command set
- Commands plus arguments/options
- Hierarchical command structure

Simple command set

- Each command is chosen to carry out a single task. The number of commands match the number of tasks.
- For small number of tasks, this can produce a system easy to learn and use.
- E.g. the vi editor of Unix (Figure 8.2, page 323)
  - H go to home position
  - L go to last line
  - M go to middle line
  - h go left one space
  - fx find the character x going forward
  - Fx find the character x going forward

Command plus arguments/options

Command plus arguments

- Follow each command by one or more arguments that indicate objects to be manipulated, e.g.
  - COPY FILEA, FILEB
  - DELETE FILEA
  - PRINT FILEA, FILEB, FILEC
- 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
  - PRINT FILEA -3, HQ

  to produce 3 copies of FILEA on the printer in the headquarters building.
- Error rates and the need for extensive training increase with the number of possible options.

Hierarchical command structure (orthogonality)

- The full set of commands is organized into a tree structure
- 5x3x4 = 60 tasks with 5 command names and 1 rule of formation

Actions	Objects	Destinations
CREATE	File	File
DISPLAY	Process	Local printer
REMOVE	Directory	Screen
COPY		Remote printer
MOVE		

The Benefits of Structure (Chap 8.4)

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

- Beneficial for
  - task concepts
  - computer concepts
  - syntactic details of command languages

Consistent Argument Ordering

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

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

What is the best?

Symbols versus Keywords

Command structure affects performance

Symbolic Editor	English-like 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

What is the best?

Hierarchical Structure and Congruence

Sources of structure that have proved advantageous include:

- Positional consistency
- Grammatical consistency
- Congruent pairing (meaningful pairs of opposites)
- Hierarchical form

What is the best?

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

Naming and Abbreviations (Chap 7.3)

There is often a lack of consistency or obvious strategy for construction of command abbreviations.

Specificity Versus Generality

Infrequent, discriminating words	insert	delete
Frequent, discriminating words	add	remove
Infrequent, non-discriminating words	amble	perceive
Frequent, non-discriminating words	walk	view
General words (frequent, non-discriminating)	alter	correct
Non-discriminating non-words (nonsense)	GAC	MIK
Discriminating non-words (icons)	abc-adbc	abc-ab

What is best? Infrequent, discriminating

What is worst? general

Familiarity of Names

- Command names chosen by designers may or may not be the ones anticipated by users
- Users can correctly guess the name chosen by designers for a function or object only about 10%-15% of the time (Furnas, 1985)
- If system allows the two most common synonyms, then hit rate goes up to 80%-90%
- Example: logout, logoff, bye

Six Potential Abbreviation Strategies

1. Simple truncation: The first, second, third, etc. letters of each command. (There is some evidence that this is the preferred abbreviation strategy for users.
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: Use with a hierarchical design plan.
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.

Command-language guidelines

- Create explicit model of objects and actions.
- Choose meaningful, specific, distinctive names.
- Try to achieve hierarchical structure.
- Provide consistent structure (hierarchy, argument order, action-object).
- Support consistent abbreviation rules (prefer truncation to one letter).
- Offer frequent users the ability to create macros.
- Consider command menus on high-speed displays.
- Limit the number of commands and ways of accomplishing a task.

NOTE: There are often trade-offs in design for different users
- Having abbreviation makes it faster to type for experts
but harder to learn for novices.