Interaction Design-2
Command Language

Matthias Rauterberg
Faculty Industrial Design
Technical University Eindhoven
g.w.m.rauterberg@tue.nl

Key references/literature-1:

chapter 1

chapter 1: introduction;
chapter 12: input and output devices.

chapter 1: what is interaction design?
chapter 2: understanding and conceptualizing interaction.
Key references/literature-2:

  chapter 7: dialog styles-command language
  chapter 6: dialog styles-question and answer
  chapter 17: user documentation.

ISO/FDIS 9241 (1997) Ergonomic requirements for office work with visual display terminals (VDTs).
  Part 15: command dialogues.

  Chapter 3: effective wording.

What is User-System Interaction about?

Goal:
Actual Performance => Desired Performance
Interface Design or Interaction Design?

user interaction system

The function-form mapping

implemented functions

perceivable user interface forms

intended semantic

perceived semantic

DMM := designer’s mental model

UMM := user’s mental model
The three important mappings

<table>
<thead>
<tr>
<th>User’s world</th>
<th>device</th>
<th>pixel world</th>
<th>semantic</th>
</tr>
</thead>
</table>
| ![User's world](image1.png) | ![Device](image2.png) | ![Pixel world](image3.png) | function-1  
function-2  
function-3  
...  
function-n |

The ‘hardware control’-’pixel world’ mapping (2)

**Input devices**
are the hardware components you use to "talk" to a computer. You use them to place requests, send messages (to the computer or to other people), move around in virtual worlds, or even shoot at "enemies" in some computer games. A few examples of commonly used input devices are:

- Computer keyboard
- Joystick
- Microphone
- Mouse
- Pen (some with, some without, a pad)
- Touch-sensitive screen
- Trackball
- TrackPoint
The ‘hardware control’-’pixel world’ mapping (2)

• **Mouse**
  
  *Advantage:* Moves cursor around the screen faster than using keystrokes.
  *Disadvantage:* Requires moving hand from keyboard to mouse and back. Repeated motion can lead to carpal tunnel syndrome.

• **Trackball**
  
  *Advantage:* Does not become too much desk space as a mouse. Is not as tiring since less motion is needed.
  *Disadvantage:* Requires fine control of the ball with just one finger or thumb. Repeated motions of the same muscles is tiring and can cause carpal tunnel syndrome.

• **Glidepad**
  
  *Advantage:* Does not need as much desk space as a mouse. Can readily be built into the keyboard. Has finer resolution. That is, to achieve the same cursor movement onscreen takes less movement of the finger on the glidepad than it does mouse movement.
  *Disadvantage:* The hand tires faster than with a mouse since there is no support. Some people don't find the motion as natural as a mouse.

• **Pen**
  
  *Advantage:* Can use handwriting instead of typing. Can use gestures instead of typing commands small size.
  *Disadvantage:* Must train device to recognize handwriting. Must learn gestures or train device to recognize the ones you create. Can lose the pen which is not usually attached to the device.

• **Touchscreen**
  
  *Advantage:* It's natural to do - reach out and touch something.
  *Disadvantage:* It's tiring if many choices must be made. It takes a lot of screen space for each choice since fingers are bigger than cursors.

• **Graphic tablet**
  
  *Advantage:* Don’t have to redraw graphics already created.
  *Disadvantage:* Expensive.

The ‘hardware control’-’pixel world’ mapping (3)

<table>
<thead>
<tr>
<th>Nature of function</th>
<th>discrete</th>
<th>continuous</th>
</tr>
</thead>
<tbody>
<tr>
<td>'button'</td>
<td>Appropriate</td>
<td>NOT appropriate</td>
</tr>
<tr>
<td></td>
<td>e.g. Cursor keys</td>
<td>e.g. Wheel control on mouse</td>
</tr>
<tr>
<td>Linear (1D)</td>
<td>possible</td>
<td>Appropriate</td>
</tr>
<tr>
<td></td>
<td>e.g. Wheel control on mouse</td>
<td></td>
</tr>
<tr>
<td>Surface (2D)</td>
<td>Possible</td>
<td>Appropriate</td>
</tr>
<tr>
<td></td>
<td>e.g. Mouse button for selection</td>
<td>e.g. Pen for handwriting, painting</td>
</tr>
<tr>
<td>Space (3D)</td>
<td>NOT appropriate</td>
<td>Appropriate</td>
</tr>
<tr>
<td></td>
<td>e.g. dataglove, datasuite</td>
<td></td>
</tr>
</tbody>
</table>
### Pros and Cons of Pointing Devices

<table>
<thead>
<tr>
<th>Pros and Cons</th>
<th>Cursor keys</th>
<th>Mouse</th>
<th>Joy stick</th>
<th>Trackball</th>
<th>Touch screen</th>
<th>Touch pad</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed</strong></td>
<td>Slow</td>
<td>Fast</td>
<td>Medium</td>
<td>Medium</td>
<td>Fast</td>
<td>medium</td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td>High</td>
<td>Medium</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Speed control</strong></td>
<td>Some</td>
<td>Yes</td>
<td>Some</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Continuous movement</strong></td>
<td>No</td>
<td>Yes</td>
<td>Soe</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Fatigue</strong></td>
<td>Low</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Directness</strong></td>
<td>Direction</td>
<td>Direction, distance, speed</td>
<td>Direction</td>
<td>Direction, distance, speed</td>
<td>Direction, distance, speed</td>
<td>Direction, distance, speed</td>
</tr>
<tr>
<td><strong>Best uses</strong></td>
<td>Cursor</td>
<td>Cursor, point, select, draw, drag</td>
<td>Cursor, point, select, track, drag</td>
<td>Cursor, point, select, track, drag</td>
<td>Point, select</td>
<td></td>
</tr>
</tbody>
</table>

### DIN 66 234 part 8 (1996)
- Suitability for the task
- Self-descriptiveness
- Conformity with user expectations
- Directness
- Error robustness

### EC directive 90/270/EEC (1990)
- Suitability for the task
- Self-descriptiveness
- Feedback about system states
- Appropriate format and pace of information presentation
- Conformity with user expectations
- Information and instruction of user
- Ease of use applicable to skill level
- Hearing and participation of users

### ISO 9241 part 10 (1996)
- Suitability for the task
- Self-descriptiveness
- Conformity with user expectations
- Suitability for learning
- Suitability for individualization
- Controllability
- Error tolerance

### Ulrich (1991)
- Task orientation
- Transparency
- Feedback
- Compatibility
- Consistency
- Support
- Selection possibilities
- User definability
- Participation
- Flexibility

(source EU Directive 90/270)  
(source Ulrich et al. 1991)
The interface architecture

(source Rauterberg, 1995, p. 17)

Three different function types

(source Rauterberg, 1995, p. 17)
The function space

What is the state-of-the-art?

- **Known interaction styles**
  - command language
  - menu
  - desktop
  - direct manipulation
What comes in the future?

- **New interaction styles**
  - speech input/output
  - computer vision based input (e.g., gestures)
  - audio interfaces (e.g., non-speech audio)
  - tactile and force feedback
  - biophysical signals (e.g., retina scanner)

Two dimensions for interaction

![Diagram showing two dimensions for interaction](source: Rauterberg, 1996)

[Feel free to have a look at: The complete history of HCI]
What is a Command Language (CL)

- CL were used by many early computers and they remain very common.
- CL may however suffer from un-meaningful names.
- CL has usually a prompt (e.g. c:> ) to tell the user where to type in the commands.
- CL requires commands to expressed using a precise syntax, and associated parameters, and are intolerant of even the slightest syntactic errors.
- The initiative of navigation is on the user who has to know what the allowable commands are and to have a clear idea of the function.
- CL is clearly unsatisfactory for novice users, but for the expert it typically represents the quickest form of communication, especially where abbreviations for commands names are allowed (e.g. <ctrl> <s> to save a file).

CL in our daily live

- Explanatory Notes to the Merriam-Webster Online Dictionary see http://www.m-w.com/help/dictnotes.htm
- Grammar described at Wikipedia see http://en.wikipedia.org/wiki/Grammar
- User Manual/Handbook [PDF]
- Meta-Syntax of a Natural Language Description
- Etc, etc, etc…
Command language interface

```
C:>  
```

[PO: perceivable object; PF: perceivable function point; DC: dialogue context]

(source Rauterberg, 1995)

Command language interaction

```
actual dialog context (DC)
representation space of the dialog and application objects (PDO + PAO)
the representational interaction point: PF = PDFIP + PAFIP
HDFIP: functional interaction point of the dialog manager
i/o interface
HAFIP: functional interaction point of the application manager
```

© M. Rauterberg, TUE
Basic definitions (1)

- **Command (C):** whole word (text string), abbreviation, or string of words representing actions requested of the system.
- **Argument (A):** independent variable (including object) used in a command phrase to modify or direct the action of a command (NOTE: arguments often include parameters).
- **Command language (CL):** command set(s), phrases, structure and syntax associated with a specific interaction of a user with a system by means of commands.
- **Command language structure:** logical structure of the command dialogue (and associated phrases).
- **Command phrase:** phrase including the command (words or their abbreviations) and associated separators and arguments (with parameters).
- **Command set:** all available commands to the user to perform a given task in a particular application context.
- **Command syntax:** sequential and other procedural requirements for inputting the components into command phrases.

Basic definitions (2)

- **Command word (name):** word (or name) used as a command in the command dialogue and representing actions requested from the system.
- **Command word abbreviation:** shortened version of a command word which is recognizable by the system as representing the command.
- **Hot keys:** keys, other than numbered function keys (e.g. F1, F2, etc.), not normally used for data entry such as modifier keys (e.g. CTRL, ALT), or key combinations (e.g. CTRL+c) which execute immediately without the need for any additional operations.
- **Keyword:** word in a command phrase identifying a particular argument class (e.g. type font).
- **Modifier:** argument that alters or limits the action of a command.
- **Parameter:** value used in conjunction with a keyword to modify the action of a command or argument.
- **Separator:** string of one or more characters, or a pause (for voice input), used to separate or organize elements in the command phrase and between command phrases.
General:
The CL should be designed such that users enter Cs in a manner which is natural or familiar to the user without concern for how the system will process the Cs to produce the output (e.g. the CL should reflect the user’s needs rather than the system process and the syntax structure should be consistent with user expectations, task requirements and input devices).

Internal consistency:
The CL should be internally consistent so Cs with the same name, function in the same way throughout the application regardless of the context. Cs that do the same thing should have the same name.

Macros:
If sequences of command words or command phrases are used frequently, users should be allowed to create and use higher level commands (macros) for these sequences.

Argument structure:
Command phrases should be be structured to minimize the complexity of arguments.

Long Lists: if arguments are long (more than 8 arguments), then additional command names should be created, functions should be combined under single arguments, or lists should be broken into some logical functional groupings

Dependencies: dependencies between arguments of a command should not dramatically change the meaning of the command phrase (e.g. command “quit filename” to save data to the file named filename; command cancel to cancel without saving {instead of the more complex “quit -c”}

Syntax:
appropriateness for modality - the syntax structure of the command phrases should be appropriate for the input modality (e.g. voice, typed input, gestures).

[EXAMPLE voice input is used exclusively and the syntax is completely consistent with spoken language]

consistency with modality - syntax should be consistent within a given modality.

[EXAMPLE for a screen-based command dialogue, the object follows the action (e.g. ‘action - object’ syntax) throughout the application]

consistency across modalities - syntax should be consistent across modalities as much as possible.

[EXAMPLE voice is used as well as typed input for commands in an application and the syntax is ‘object-action’ for both modalities]

Command separation:
if the input of multiple commands is allowed, a simple and consistent method to separate commands should be used.

[EXAMPLE BLANKS - if system constrains do not require the use of a specific separator, BLANKS should be used rather than punctuation marks to separate commands]

[EXAMPLE STANDARD SYMBOL - if system constrains require a separator other than BLANKS to distinguish separate stacked commands, a simple STANDRAD SYMBOL should be used consistently; e.g. using the slash (/) in the sequence of commands words “sort|format|print”]
Syntactical structure (3)

- **Language compatibility**
  command structure (semantics and syntax) should correspond to the terminology and data organisation familiar or natural to the user.
  [EXAMPLE: the rules for natural language syntax (e.g. English, French, etc) are applied in designing a query language]
- **Command arguments**
  command arguments should be easy for the user to specify and to relate to the commands that they modify.
  [NOTE: in some cases, it may be appropriate to represent arguments as names rather than single letters]
  command elements linkage - the command dialogue should be structured so that the relationship between the command phrase elements is clear.
  arguments format - if appropriate to the task, keyword format (parameter designated by argument identifiers that precede them) should be used rather than positional formats (parameter designated by their sequential position in the argument string following the command).
  [EXAMPLE (keyword format): `change shape=round colour=red size=4`]
  [EXAMPLE (positional format): `change round red 4`]
  placement of optional argument - if keyword format is used, optional arguments should be placed at the end of the argument list.
  Separation of arguments - (a) if BLANK spaces are allowed, a variable number of blanks should be allowed between command elements; (b) if other separators are used, a simple standard symbol should be used consistently.
  [EXAMPLE: using the comma (,) in the command phrase “`print fileA,fileB,fileC`”]

Syntactical structure (4)

- **FLAT vs DEEP structure:**
  a flat CL has lots of commands at every level whereas a deep CL has a few basic commands which put the system into a level of input mode at which it will recognize certain commands.
  Advantage of a flat CLs is that they are very powerful (very complex command sequences can be expressed by stringing the right commands together, but require users to remember lots of command words (e.g. UNIX).
- **POSITIONAL vs KEYWORD:**
  positional CLs have a strict syntax: the order in which command words are issued contains information (e.g. UNIX command: `mv filename directoryname`). For CLs which recognize keywords, the user can string commands together in any order - if there is potential ambiguity, the system will query/ask for confirmation before executing a command.
- **MNEMONICS:**
  most CLs use abbreviated natural language words as commands to facilitate recall (e.g. `mkdir` for “make directory”).
Command representation (1)

- **Command names**
  - **general**: command names should be easily related to their function, generally stated as verbs (usually in imperative form), be easily remembered by users, and be consistent with the user’s task requirements, experience and language usage.
  - **distinctiveness**: (a) command names should be distinctive; (b) command names should be avoided that look or sound similar but have different meanings; (c) if command operations have inverse or counterparts, congruent pairs of commands for these operations should be provided.
  - [EXAMPLE (a) in English, the words *insert* and *delete* are more semantically distinct than *add* and *remove* (i.e. *add* and *remove* typically have many different interpretations)]
  - [EXAMPLE (b) use *replace* rather than *change*]
  - [EXAMPLE (b) in English, *store* and *restore* should be avoided because they have different meanings but sound similar]
  - **user orientation**: command names should be chosen that are consistent with the user’s experience and correspond to the user’s operational language.
  - [NOTE if there are multiple user groups, it may be important to provide different sets of command names for these different groups]
  - **emotional content**: words selected as command words should be emotionally neutral.
  - [EXAMPLE in English use *cancel* instead of *abort* and use *delete* rather than *kill*]
  - **command word length**: if command input is typed, command words should not exceed 7 characters.
  - **suffixes and prefixes**: command word should not incorporate unnecessary suffixes or prefixes.
  - [EXAMPLE in English, *delete* rather than *deleting, deleted, or deletes*]

Command representation (2)

- **Abbreviations**
  - **general**: if users must type commands, they should be able to use abbreviations instead of typing complete commands. If it is appropriate to the task to provide command abbreviations, these abbreviations should be obvious to the user, easily remembered, and facilitate command input.
  - [NOTE if the command input is an abbreviation and system constrains allow, the ‘whole’ command name may be displayed prior to, or simultaneous with, execution (especially during learning the CL)]
  - **abbreviations rules**: (a) if command names are shortened, they should be shortened using as simple a rule as possible; that rule should apply to all commands and those arguments that can be abbreviated;
  - (b) if the task requires the user to generate and remember commands, simple truncation should be used to shorten commands.
  - [EXAMPLE (a) truncation: *pr* for *print*; dropping of vowels: *print for print*]
  - [EXAMPLE (b) to drop off characters beyond those necessary to keep the command unique (e.g. *q* for *quit, qui* for *quit* and *gue* for *query*)]
Command representation (3)

- **Function keys and hot keys**
  - **general** - if function keys or hot keys are used for command input, their use should be obvious to users or the key assignments should be readily accessible and these assignments should be consistent throughout the application.
  - **NOTE** consider using function keys and hot keys for frequently used commands or when it is important to speed up command entry.
  - **function key consistency** - function key assignments for commands should be consistent across related tasks within an application, particularly for ‘generic’ commands like help.
  - **hot key consistency** - hot keys should have the same meaning throughout the whole application.
  - **NOTE** if commands can be accessed by menu as well as typing, the hot key assignments should be the same as the accelerators used in the menus.
  - **consistent grouping of modifiers** - if modifier keys (e.g. **CTRL** or **ALT** keys) are used with other keys, there should be a consistent rule of the modifier key usage.
  - **EXAMPLE** **ALT**+ letter keys is used for navigation and window manipulation and **CTRL**+ other letter keys is used for data manipulation.
  - **limited modifiers** - multiple simultaneous modifier keys should be used in hot keys only if there are more commands than can be accommodated meaningfully by single modifier keys.
  - **EXAMPLE** in a dialogue, **ALT**+p (rather than **ALT**+CTRL+p) is used to issue a print command.
  - **NOTE** if possible, use letter keys that are mnemonic in combination with modifiers; it may be desirable to require the depression of more than one modifier key to reduce the possibility of accidentally causing a destructive action.

Command name design (1)

<table>
<thead>
<tr>
<th>Congruent: Hierarchical</th>
<th>Congruent: Non hierarchical</th>
<th>Non congruent: Hierarchical</th>
<th>Non congruent: Non hierarchical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move robot forward</td>
<td>Advance</td>
<td>Move robot forward</td>
<td>Go</td>
</tr>
<tr>
<td>move robot backward</td>
<td>retreat</td>
<td>change robot backward</td>
<td>back</td>
</tr>
<tr>
<td>Move robot right</td>
<td>Right</td>
<td>Change robot right</td>
<td>Turn</td>
</tr>
<tr>
<td>move robot left</td>
<td>left</td>
<td>move robot left</td>
<td>left</td>
</tr>
<tr>
<td>Move robot up</td>
<td>Straighten</td>
<td>Change robot up</td>
<td>Up</td>
</tr>
<tr>
<td>move robot down</td>
<td>bend</td>
<td>move robot down</td>
<td>bend</td>
</tr>
<tr>
<td>Move arm right</td>
<td>Swing out</td>
<td>Change arm right</td>
<td>Pivot</td>
</tr>
<tr>
<td>move arm left</td>
<td>swing in</td>
<td>move arm left</td>
<td>sweep</td>
</tr>
<tr>
<td>Change arm open</td>
<td>Release</td>
<td>Change arm open</td>
<td>Unhook</td>
</tr>
<tr>
<td>change arm closed</td>
<td>take</td>
<td>move arm close</td>
<td>grab</td>
</tr>
</tbody>
</table>

**Rating:** 1.86 1.63 1.81 2.73
**Errors #1:** 0.50 2.13 4.25 1.63
**Errors #2:** 0.75 1.25 4.63 2.38

Command name design (2)

<table>
<thead>
<tr>
<th>Notational</th>
<th>English-like</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find:/tooth/-1</td>
<td>Backward to “tooth”</td>
</tr>
<tr>
<td>List:10</td>
<td>List 10 lines</td>
</tr>
<tr>
<td>List:/ko/;*</td>
<td>List all lines with “ko”</td>
</tr>
<tr>
<td>l:/ko/;*</td>
<td>lawl “ko”</td>
</tr>
</tbody>
</table>

% task completed: novice 28 42
% task completed: familiar 43 62
% task completed: expert 74 84
% erroneous commands: novice 19.0 11.0
% erroneous commands: familiar 18.0 6.4
% erroneous commands: expert 9.9 5.6


Command name design (3)

<table>
<thead>
<tr>
<th>Specific</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Transfer</td>
</tr>
<tr>
<td>Send</td>
<td>Put</td>
</tr>
<tr>
<td>Front</td>
<td>Move</td>
</tr>
<tr>
<td>Insert</td>
<td>Add</td>
</tr>
<tr>
<td>Rubout</td>
<td>Edit</td>
</tr>
<tr>
<td>Split</td>
<td>Open</td>
</tr>
</tbody>
</table>

Command language syntax design

**Poor:**

VolB!FileA!D$$

FileA!VolB!ER$L!:KO!:!*$$

**Improved:**

search (for) filea (in) volb.

open filea (in) volb. list all lines with "KO".  

OR

s filea volb.

o filea volb. lal "KO".

---

Command name abbreviation design (1)

1. **Truncation:**

append = APPE
execute = EXEC

2. **Vowel deletion:**

append = APPND
execute = EXCT

3. **Contraction:**

append = APND
execute = EXTE

4. **Minimum to distinguish:**

translate = TRANS
transfer = TRANF

5. **Phonetic:**

append = APND
execute = XQT

6. **First syllable:**

append = APPN
execute = EXEC

7. **User created:**

(by individuals)
# Command name abbreviation design (2)

<table>
<thead>
<tr>
<th>Name</th>
<th>Abbreviations Poor</th>
<th>Abbreviations Improved</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Move forward</strong></td>
<td>MovF</td>
<td>MovF</td>
</tr>
<tr>
<td><strong>Move backward</strong></td>
<td>Mvb</td>
<td>Mvb</td>
</tr>
<tr>
<td><strong>Insert</strong></td>
<td>I</td>
<td>Ins</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>Dl</td>
<td>Del</td>
</tr>
<tr>
<td><strong>Replace</strong></td>
<td>Repl</td>
<td>Rep</td>
</tr>
<tr>
<td><strong>Search</strong></td>
<td>Srch</td>
<td>Sea</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>X</td>
<td>Del</td>
</tr>
<tr>
<td><strong>Send</strong></td>
<td>So</td>
<td>Sen</td>
</tr>
<tr>
<td><strong>Print</strong></td>
<td>Prt</td>
<td>Pri</td>
</tr>
<tr>
<td><strong>Search</strong></td>
<td>Srch</td>
<td>Sea</td>
</tr>
<tr>
<td><strong>Send</strong></td>
<td>Sn</td>
<td>Sen</td>
</tr>
<tr>
<td><strong>Find</strong></td>
<td>Fi</td>
<td>Fin</td>
</tr>
<tr>
<td><strong>Choose</strong></td>
<td>Ch</td>
<td>Cho</td>
</tr>
</tbody>
</table>

[Source: Mayhew (1992) chap 7]

# Command name abbreviation design (3)

<table>
<thead>
<tr>
<th>Name</th>
<th>Abbreviations Poor</th>
<th>Abbreviations Improved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Next page</td>
<td>NP</td>
<td>NP</td>
</tr>
<tr>
<td>Previous page</td>
<td>PP</td>
<td>PP</td>
</tr>
<tr>
<td>Home</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>Next document</td>
<td>ND</td>
<td>ND</td>
</tr>
<tr>
<td>Previous document</td>
<td>PD</td>
<td>PD</td>
</tr>
<tr>
<td>First document</td>
<td>FD</td>
<td>FD</td>
</tr>
<tr>
<td>Screen print</td>
<td>SP</td>
<td>SP</td>
</tr>
<tr>
<td>Print document</td>
<td>PR</td>
<td>PR</td>
</tr>
<tr>
<td>Mail it</td>
<td>MI</td>
<td>MI</td>
</tr>
<tr>
<td>Request</td>
<td>R</td>
<td>DR</td>
</tr>
<tr>
<td>Help</td>
<td>H</td>
<td>DH</td>
</tr>
<tr>
<td>Time</td>
<td>T</td>
<td>DT</td>
</tr>
<tr>
<td>Full</td>
<td>FL</td>
<td>FF</td>
</tr>
<tr>
<td>Short</td>
<td>SH</td>
<td>FS</td>
</tr>
<tr>
<td>Location</td>
<td>LO</td>
<td>FL</td>
</tr>
<tr>
<td>Select services</td>
<td>SRT</td>
<td>SS</td>
</tr>
<tr>
<td>Exit</td>
<td>EX</td>
<td>ES</td>
</tr>
<tr>
<td>Modify</td>
<td>M</td>
<td>MR</td>
</tr>
<tr>
<td>New search</td>
<td>NR</td>
<td>CR</td>
</tr>
<tr>
<td>Change drawer</td>
<td>CD</td>
<td>CD</td>
</tr>
<tr>
<td>Change cabinet</td>
<td>CC</td>
<td>CC</td>
</tr>
</tbody>
</table>

[Source: Mayhew (1992) chap 7]
Input and output considerations (1)

- **General**
  
  Users should be in control of the dialogue at all times, be able to easily recover from errors, and not be required to input more information than is necessary for successful task performance.

- **Command reuse**
  
  If the same sets of commands are used repeatedly during a work session, the system should provide a way of reusing the commands without requiring the user to type them again. 
  
  **EXAMPLE** giving users a command history list from which they can select a previously used command

- **Command queuing**
  
  Users should be provided with the capability to key in a series of commands (command queuing or stacking) rather than wait for the system to execute each individual command. 
  
  **NOTE** separators should be provided to separate command strings (see above under separators)

- **Error correction**
  
  If errors occur, re-entry, or editing, should be required preferably for the erroneous portion of the command and associated parameters.

- **Editing**
  
  (a) Users should be allowed to edit commands prior to execution; (b) If the application has a text editor, the same text editing conventions used in the text editor should be apply to command dialogue editing.

- **Misspelling**
  
  If appropriate for the task and system constraints allow, the system should provide for interpretation and acceptance of misspelled commands unless there is ambiguity as to what command was intended.

Input and output considerations (2)

- **Defaults**
  
  Defaults should be provided to minimise typing requirements and to facilitate learning. 
  
  **EXAMPLE** If the disk drive is not identified it is assumed to be the currently set default drive
  
  **NOTE** Arguments that have default parameter values are often referred to as optional arguments

- **Destructive commands**
  
  (a) If a command may have unintentional or destructive consequences (e.g. delete a file); the user should be allowed to cancel or undo the previous (last) command and its effects; (b) The user should be required to confirm the intention of the command before command execution.

- **Customisation**
  
  If system constraints allow, users should have the capability to designate and use synonyms for commands and command macros and they should be able to revert back to the default names when desired.

- **Echoing typed commands**
  
  (a) The user’s input should be displayed (echoed) in a consistent position; (b) typed in command characters should be displayed (echoed) as the user types each character. 
  
  **EXAMPLE** (a) displayed on a ‘command line’ at the bottom of the screen or displayed after the prompt on the screen

- **Output control**
  
  If appropriate to the task and system constraints allow, the command phrase should allow arguments for redirecting output, interrupting output, or stopping output.

- **Consistent output format**
  
  Commands resulting in similar or related output should present their resulting data in a consistent format. 
  
  **EXAMPLE** Use of a single presentation format for lists of files, processes, directories, etc.
Feedback and Help (1)

• **General**
  Feedback and help should be provide users with information allowing them to control the dialogue, recognise and recover from errors, and determine their next course of action.

• **Command processing**
  - **Completion** - the system should indicate that the command processing has been completed by displaying the output resulting from the command and/or prompt for the next command.
  - **Intermediate feedback** - if the command processing is expected to continue for a longer period (more than 5 seconds), visual feedback indicating that the process is continuing should be provided to the user.
  - **Processing status** - if appropriate to the task and system constrains allow, user should be provided with feedback concerning the relative amount of time remaining to complete the process.

• **Error feedback**
  - **Timing** - error feedback should be provided after the full command (including associated parameters) has been entered rather than as soon as the error is discovered by the system.
  - **Highlighting** - the unacceptable portion of the command should be highlighted (in the context of the full command or a logical part thereof).

Feedback and Help (2)

• **Command information**
  If appropriate to the task, the user should be provided on request with information on:
  * commands available and their meaning
  * appropriate syntax structure
  * required and optional arguments available (especially if the number is large)
  * command entry history

• **Performance aids**
  Performance aids should be provided depicting command characteristics (e.g. name, function, syntax, parameters, abbreviations, hot key, function key assignment).

• **Long argument lists**
  If a command has a long list of arguments and associated parameters, the use of additional dialogue techniques should be provided.
How to describe a CL?

### Content

**Meta Syntax**
Syntax to describe the syntax of the command language with meta symbols

**CL-Syntax**
Syntax of the command language; major distinction between non-terminal and terminal symbols

**Commands**
List of all specified commands

### Examples

- Backus-Naur-Form (BNF)
- Syntax diagram
- State-transition net
- Written text

`<Commands> ::= <name> [arguments] [parameters]*`

- `cancel`
- `print filename`
- `list commands`

---

What is a ‘meta syntax’?

A **meta syntax** is described e.g. in Backus-Naur-Form (BNF)

BNF was first used by John Backus and Peter Naur to describe Algol in 1963. It consists of a set of rules or productions of the form:

- `<syntactic category> ::=` a sequence of terminals, syntactic categories, and the symbol `|`

Definition of all used meta-symbols:

- ` ::= "is defined as"`
- juxtaposition means concatenation ;
- `|` means alternation (the logical exclusive OR) ;
- optional items are enclosed in meta symbols `{ and }`
- repetitive items (zero or more times) are enclosed in meta symbols `{ and }`
- parentheses merely serve for grouping, i.e., `( a | b ) c` stands for: `a c | b c`
- **non-terminals** are necessary to describe the syntactical structure and are surrounded by meta symbols `< and >`
- **terminals** describe the atomic operations (e.g. commands) (terminals of only one character are surrounded by quotes “a”)
BNF is a ‘meta syntax’

Now as an example (maybe not the easiest to read, but the different colors help!)

here is the definition of the syntax of BNF expressed in BNF (see also Wirth’s BNF):

```
<bnf-syntax> ::= { <bnf-rule> }
<bnf-rule> ::= <symbol-1> <identifier> <symbol-2> <symbol-3> <expression>
<expression> ::= <term> { "|" <term> }
<term> ::= <factor> { "|" <factor> }
<factor> ::= <identifier> | <quoted-symbol> | ( "expression" )
               | [ "expression" ] | { "expression" }
<identifier> ::= <letter> { <letter> | <digit> }
<quote>
<letter> ::= "a" | "b" | "c" | ... | "z" | "+" | "-"
<digit> ::= Caterpillar "O" | "1" | "2" | ... | "9"
<symbol-1> ::= "<" <symbol-2> ::= ">
<symbol-3> ::= "":="
<symbol> ::= ( "" ) | [ "" ] | { "" }
```

All blue items are meta-symbols, all red items are non-terminals to describe the syntactical structure, and all black items are terminals (“atomic building blocks”).

BNF is not only important to describe syntax rules using non-terminals and terminals, but it is also very commonly used (with variants, e.g. EBNF) by syntactic tools.

EBNF is another ‘meta syntax’

A meta syntax described in Extended Backus-Naur-Form (EBNF) [see also UNIX as a concrete example]

The following meta-symbols are added to BNF to describe the syntax of a programming language relying heavily on recursion to provide lists of items. EBNF uses these extensions:

1) Parentheses ( <item1> <item2> ) are used for grouping of items.
2) <item> ? or [ <item> ] means item is optional.
3) <item> * or { <item> } means to take zero or more occurrences of item.
4) <item> + means to take one or more occurrences of item.
5) Bold item means terminal symbol of the defined syntax.
6) Italic item means placeholder or variable to be filled in later in the concrete context of use.

Example
```
<method declaration> ::= <method modifiers>? <result type> <method declarator> <throws>? <method body>
<method modifiers> ::= <public> | <protected> | <private> | <static> | <abstract> | <final> | <synchronized> | <native>
```

© M. Rauterberg, TUE 46/49
How to define a ‘command language’?

The command language SIMPLE can be defined by using EBNF. Here is the definition of the syntax of SIMPLE plus commands defined with EBNF:

```plaintext
<simple> ::= <prompt> <command> [<command_separator> <command>]*
<command> ::= <command_name> [ ' ' <argument> [ ' ' <parameter> ] ]
<argument> ::= <letter> [ <letter> | <digit> ]
<parameter> ::= <letter> [ <letter> | <digit> ]
<prompt> ::= (SIMPLE)
<command_separator> ::= """
<command_name> ::= (CANCEL | CA | UNDO | UN | WRITE | WR | COPY | CO | DELETE | DE)

WRITE filename [length]
COPY input-filename [copies] output-filename
...
```

You can copy this general syntax definition and adapt it to your specific command language with your specific set of commands. You do not have to use different colors.

When to use CLs?

- **User characteristics**
  - positive attitude
  - high motivation
- **Knowledge and expertise**
  - moderate to high typing skills
  - high system experiences
  - high task experiences
  - high application experiences
  - infrequent use of other interaction styles
  - high computer literacy
- **Job and task characteristics**
  - high frequency of use
  - formal training
  - mandatory use
  - low turnover rate
  - high task importance
  - low task structure
About HCI in general:


About design principles:

- C. Green, G. Cockton (eds.): Design principles for interactive software. Capman & Hall, 1996.

About usability evaluation methods:


About Design: