Using Guidelines

- guidelines provide us with advice to the solution of design problems, and suggest possible strategies
- they are *heuristics*, that is they are assumptions derived from past experience, or on empirical research
- be critical: guidelines can serve many purposes, but also have limitations
- roles of guidelines:
  - raising awareness of concepts, when dealing with unfamiliar problems
  - assisting in design choices (e.g. where to locate the command line?)
  - offering overall strategies, so that consistency is improved
  - supporting evaluation, by providing checklists
- limitations of guidelines:
  - which guideline to select?
  - how to apply a guideline?
  - what if two, constrasting guidelines apply?

The 8 Golden Rules of Interface Design

1. Strive for consistency
2. Take expert users into consideration
3. Offer informative feedback
4. Design dialogs to yield closure
5. Offer error prevention and simple error handling
6. Permit easy reversal of actions
7. Support internal locus of control
8. Reduce short-term memory load

Consistency

- insuring consistency is the most important guideline, and the most frequently violated
- consistency means:
  - similar sequences of actions in similar situations
  - identical terminology in prompts, menus and help screens
  - consistent colour, layout, capitalization, fonts etc.
  - if exception $\rightarrow$ comprehensible and limited (e.g. no echoing of passwords)
- consistency across:
  - applications of the same environment
  - versions of the same system
  - computer system and real world system (whenever possible)

An Example: Defining a Command Grammar

A	B	C
delete/insert char	delete/insert char	delete/insert char
delete/insert word	remove/bring word	remove/insert word
delete/insert line	destroy/create line	delete/insert line
delete/insert par	kill/birth par	delete/insert par

- A is consistent all over
- B lacks on any attempt at consistency: it is not “wrong”, but requires a greater learning effort
- C violates consistency only once... but you have to remember which the different case is!

Use a Technique

- again, a good task analysis can help
- once the full set of task-actions mappings is written down, derive from the dictionary of tasks a consistent grammar
- for example: a grammar for cursor control would have a dictionary of tasks:
  1. move-cursor-one-char-forward (Direction=forward; Unit=char)
  2. move-cursor-one-char-backward (Direction=backward; Unit=char)
  3. move-cursor-one-word-forward (Direction=forward; Unit=word)
  4. move-cursor-one-word-backward (Direction=backward; Unit=word)

- the high level rules for the grammar can be described as follows:
  1. task [Direction,Unit] → symbol[Direction] + letter[Unit]
  2. symbol[Direction = forward] → "CTRL"
  3. symbol[Direction = backward] → "ALT"
  4. letter[Unit = char] → "C"
  5. letter[Unit = word] → "W"
- resulting consistent grammar:
  1. move-cursor-one-char-forward: CTRL-C
  2. move-cursor-one-char-backward: ALT-C
  3. move-cursor-one-word-forward: CTRL-W
  4. move-cursor-one-word-backward: ALT-W

- an alternative grammar:
  1. task [Unit,Direction] → symbol[Unit] + symbol[Direction]
  2. symbol[Unit = char] → "CTRL"
  3. symbol[Unit = word] → "ALT"
  4. symbol[Direction = forward] → "→"
  5. symbol[Direction = backward] → "←"
- resulting consistent grammar:
  1. move-cursor-one-char-forward: CTRL-→
  2. move-cursor-one-char-backward: CTRL-←
  3. move-cursor-one-word-forward: ALT-→
  4. move-cursor-one-word-backward: ALT-←

Closure

- "relief" caused by having finished a task (e.g. looking up and dialling a telephone number)
- if humans know that a task has been accomplished, they feel free to move to the subsequent task
- a message can be used to create a closure condition (e.g. a welcome after a successful login)
- closure helps people chunking information, so that the whole task can be remembered as a unique chunk as soon as the user becomes more expert

When closure is a problem

Task: deleting a paragraph by using the “delete” button:

```
Method for deleting highlighted text
   Step 1. move mouse on delete button
   Step 2. press mouse button
   Step 3. observe button highlighted
   Step 4. release mouse button
   Step 5. observe remove highlight
   Step 6. observe text deleted
   Step 7. report goal accomplished
```

- However: if the mouse slips from the button, the click is ignored:
  - the button behaviour is identical
  - but the text will NOT be deleted
- Problem: two cognitive steps (5 and 6) to perform in sequence.
- Users will typically reach closure after step 5 (especially if expert!!!), and won’t realise missing the button.
- Solution: force closure on step 6 (e.g. sound alert), or avoid problem altogether (“magnetic” button, difficult to move out of).

When is locus of control NOT on the user?

- when the user thinks something can be done, but in fact this is not true
- when the user thinks something can be done IN A PARTICULAR WAY, but in fact this is not true
- when the user is forced to perform unnecessary actions
- when the user makes a mistake that could have been avoided
- when a task is unnecessarily hard
- when the system does something the user would not expect

IN GENERAL:

> When the user is forced to any waste of time because of some interface feature, and not because of the task itself the user intends to perform.

Users should be *initiators* of actions, not *responders*.

Locus of control

- the controls represent the means by which the user communicates with the application. The quality of the communication depends on two aspects of the controls:
  - the appropriateness of the control for the task
  - the consistency of the rules under which the control operates.
- select the wrong tool for the job, or change the rules under which the tool operates, and you will create problems for your users.
- users should feel they are in charge