

Design Principles and Usability Heuristics

“Introspective” evaluation method to use to inspect an interface for usability problems.

Also good for to consider when trying to avoid common design pitfalls .

Design principles and usability heuristics (I)

The design principles represent a broad set of general rules based on research and experience that also describe features of “usable” systems.

- broad usability statements that guide a developer’s design efforts
- derived by evaluating common design problems across many systems

Heuristic evaluation

- Take these same principles and use them to “evaluate” a system for usability problems.
- Reasonably popular approach since actual user involvement is not required (cheaper and logistically easier) and it end up catching many design flaws.
- Is considered an “expert review” technique.

Design principles and usability heuristics (II)

Advantages

- It is a “minimalist” approach where using a few general guidelines grounded in research and experience can help identify and correct the majority of usability problems.
 - Also, easily remembered list and easily applied with modest effort.
- Considered “discount usability engineering” due to its relative low cost and the speed at which it can be applied. However, the experience level of the evaluators has an impact, so often done by usability experts.

Evan Golub / Ben Bederson / Saul Greenberg

Design principles and usability heuristics (III)

Challenges (for lack of a better word)

- These principles can’t be treated as a simple checklist.
 - Note: “If done wrong, that’s bad” is a common “disadvantage”, but it is worth noting here.
- There are subtleties involved in their use and in classifying some specific issues that are raised.
- Some consider this a stage before “real” user testing to catch many issues before users are brought in for usability/performance testing

Evan Golub / Ben Bederson / Saul Greenberg

Why is “discount usability engineering” approach?

Relative to user observational studies, this is cheap/fast/easy which can be critical in today’s product cycle...

- There are no special labs or equipment needed.
 - For many things, likely able to run it on your own machine in your office
 - Interesting bonus: can even be used on *paper prototypes*
- Doing this type of evaluation can be done on the order of one day where other usability testing could take weeks.
- Once the approach is understood by a team it can be used in many scenarios with little additional learning and the more careful you are, the better the results get.

Evan Golub / Ben Bederson / Saul Greenberg

Heuristic Evaluation

Developed by Jakob Nielsen (1990)

- Original list of heuristics seems inspired by Shneiderman’s “Eight Golden Rules” of design.
- Nielsen has had multiple similar lists over the years.
- Jill Gerhardt-Powals has a list as well but they have a very different feel to them.

Helps find usability problems in a UI design

Small set (3-5) of evaluators examine UI

- Independently check for compliance with usability principles
- Evaluators only communicate after they are done with their eval and the findings are then aggregated.
- Common for overlap but in places different evaluators will find or identify different problems

Evan Golub / Ben Bederson / Saul Greenberg

Heuristic Evaluation Process

Evaluators go through UI several times

- inspects various dialogue elements
- compares with list of usability principles
- consider other principles/results that come to mind

Usability principles

- Nielsen's "heuristics"
 - there are several slightly different sets (we will see one) of heuristics
- supplementary list of category-specific heuristics
 - competitive analysis & user testing of existing products

Use violations to redesign/fix problems

Evan Golub / Ben Bederson / Saul Greenberg

Phases of Heuristic Evaluation

1) Pre-evaluation training

- give evaluators needed domain knowledge and information on the scenario

2) Evaluation

- individuals evaluate and then aggregate results

3) Severity rating

- determine how severe each problem is (priority)

4) Debriefing

- discuss the outcome with design team

Evan Golub / Ben Bederson / Saul Greenberg

How to Perform Evaluation

At least two passes for each evaluator

- first to get feel for flow and scope of system
- second to focus on specific elements

If system is walk-up-and-use or evaluators are domain experts, then no assistance needed

- otherwise might supply evaluators with scenarios

Each evaluator produces list of problems

- explain why with reference to heuristic or other info.
- be specific and list each problem separately

Evan Golub / Ben Bederson / Saul Greenberg

Examples

Can't copy info from one window to another

- violates "Minimize the users' memory load"
- fix: allow copying

Typography uses mix of upper/lower case formats and fonts

- violates "Consistency and standards"
- slows users down
- probably wouldn't be found by user testing
- fix: pick a single format for entire interface

Evan Golub / Ben Bederson / Saul Greenberg

Severity Rating

Used to allocate resources to fix problems

Estimates of need for more usability efforts

Combination of

- frequency
- impact
- persistence (one time or repeating)

Should be calculated after all evaluations are in

Should be done independently by all judges

Evan Golub / Ben Bederson / Saul Greenberg

Nielsen's Example Ratings List

0 = I don't agree that this is a usability problem at all.

1 = Cosmetic problem only.

need not be fixed unless extra time is available on project

2 = Minor usability problem.

fixing this should be given low priority

3 = Major usability problem.

important to fix, so should be given high priority

4 = Usability catastrophe.

imperative to fix this before product can be released

Some comments on the above...

- Although Nielsen provides a "0" rating, it is unclear where it would be used
 - perhaps on a "second opinion" evaluation
- It is possible for a cosmetic problem to be a usability catastrophe
 - imagine a green checkmark meaning "bad/danger"

Evan Golub / Ben Bederson / Saul Greenberg

Debriefing

Conduct with evaluators, observers, and development team members

Discuss general characteristics of UI

Suggest potential improvements to address major usability problems

Development team rates how hard things are to fix

Make it a brainstorming session

- little criticism until end of session

Evan Golub / Ben Bederson / Saul Greenberg

Results of Using HE

Discount: benefit-cost ratio of 48 [Nielsen94]

- cost was \$10,500 for benefit of \$500,000
- value of each problem ~15K (Nielsen & Landauer)
- how might we calculate this value?
 - in-house → productivity
 - open market → sales

Correlation between severity & finding w/ HE

http://www.useit.com/papers/heuristic/heuristic_evaluation.html

Evan Golub / Ben Bederson / Saul Greenberg

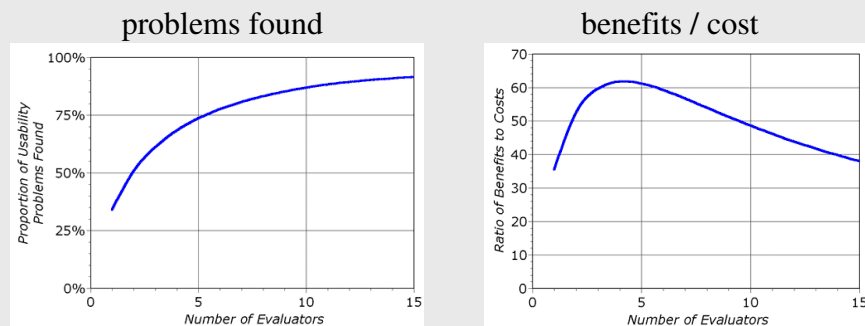
Why Multiple Evaluators?

Single evaluator achieves poor results

- Previous comparisons indicated that a single evaluator will only find around 35% of the actual usability problems but that using five evaluators will lead to finding around 75% of them (some don't get found until the users try it out).
- One question that came up was “why not more evaluators?” Would it help to go up to 10? 20?
 - The reality is that adding evaluators costs more (not just scaling for number of people but also increased time for everyone during the aggregation stage).
 - Having that many evaluators won't identify many more problems in practice.

Evan Golub / Ben Bederson / Saul Greenberg

Why Multiple Evaluators (cont)?



(graphs for a specific example study that was done)

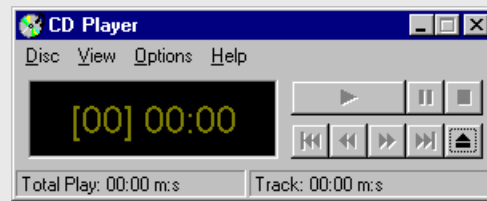
Evan Golub / Ben Bederson / Saul Greenberg

1 Simple and natural dialogue

Conform to the user's conceptual model.

Match the users' task in as natural a way as possible

- maximize mapping between interface and task semantics



Good? Bad?

This has changed over time as people went away from audio tape in their lives...

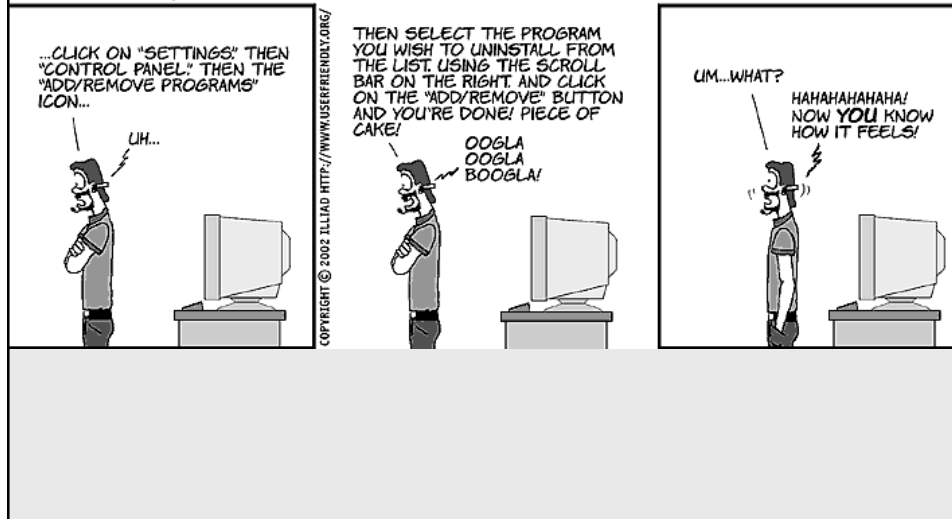
1 Simple and natural dialogue

Present exactly the information the user needs.

- less is more
 - less to learn, to get wrong, to distract...
- information should appear in natural order
 - related information is graphically clustered
 - order of accessing information matches user's expectations
- remove or hide irrelevant or rarely needed information
 - competes with important information on screen
- use windows frugally
 - don't make navigation and window management excessively complex

2 Speak the users' language

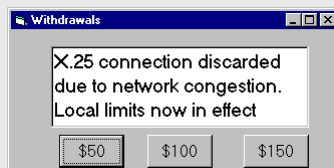
USER FRIENDLY by Illiad



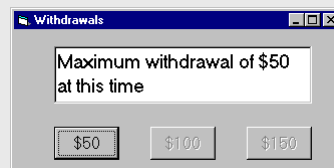
2 Speak the users' language

Use terminology based on users' language for task.

- e.g. withdrawing money from a bank machine



Bad



Better

Use meaningful mnemonics, icons, and abbreviations.

- eg: File / Save
 - Ctrl + S (abbreviation)
 - Alt F S (mnemonic for menu action)
 - Open folder (tooltip icon)



NOTE: This could fall under #7 providing shortcuts.

2 Speak the users' language

Ex: Consider a virus detection program that may have to be occasionally turned off.

One option would be to have an “override mode” that when activated would turn off the virus detection.

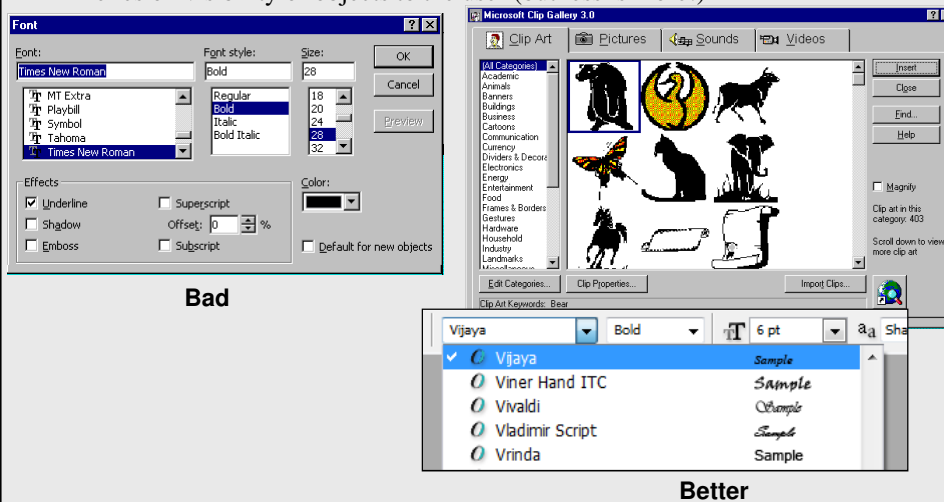
But this would be *on* when the user wanted the utility to be *off* – conflicting with the users' model

Alternatively, a checkbox that was on when the utility would be on would speak the users' language.

3 Minimize user's memory load

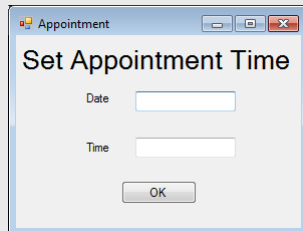
Promote recognition over recall.

- computers are good at remembering thing, people not as much...
- menus, icons, choice dialog boxes vs command lines, field formats
- relies on visibility of objects to the user (but less is more!)



3: Minimize user's memory load

Describe required input format and provide an example or a default or a selection interface.



Appointment

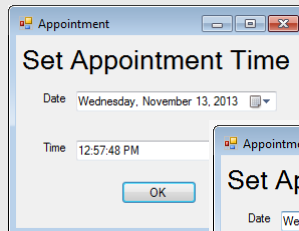
Set Appointment Time

Date

Time

OK

Bad



Appointment

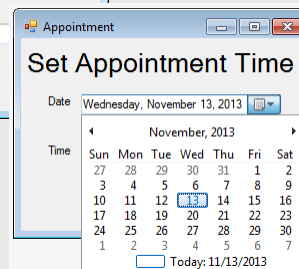
Set Appointment Time

Date Wednesday, November 13, 2013

Time 12:57:48 PM

OK

Better



Appointment

Set Appointment Time

Date Wednesday, November 13, 2013

Time

| November, 2013 | | | | | | |
|----------------|-----|-----|-----|-----|-----|-----|
| Sun | Mon | Tue | Wed | Thu | Fri | Sat |
| 27 | 28 | 29 | 30 | 31 | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Today: 11/13/2013

Small number of rules applied universally.

generic commands

- same command can be applied to all interface objects
 - *interpreted in context of interface object*
- copy, cut, paste, drag 'n drop, ... for characters, words, paragraphs, circles, files

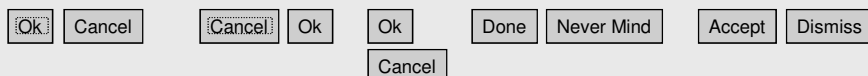
4: Be consistent

Consistency of effects.

- same words, commands, actions will always have the same effect in equivalent situations
 - predictability

Consistency of language and graphics.

- same information/controls in same location on all screens / dialog boxes



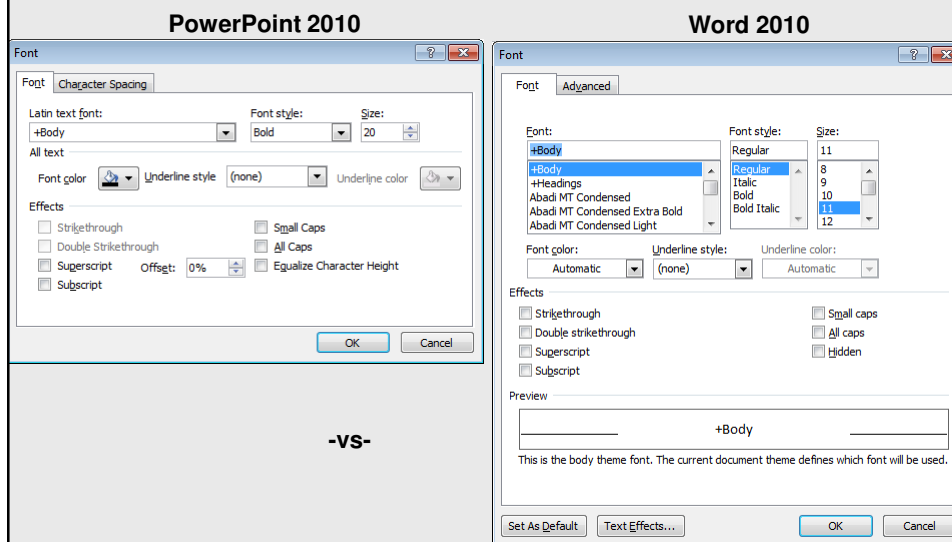
- forms follow boiler plate
- same visual appearance across the system (e.g. widgets)
 - e.g. different scroll bars in a single window system!

Consistency of input.

- consistent syntax across complete system

4: Be consistent

In application suites, have individual applications consistent with the other individual applications in the suite.



4: Be consistent

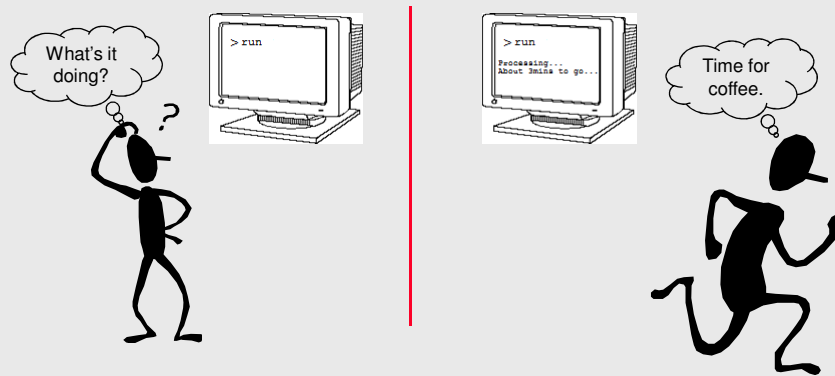
In application suites, have individual applications consistent with the other individual applications in the suite.



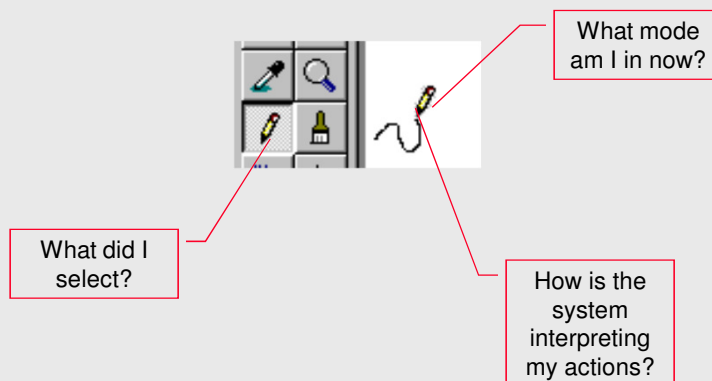
5: Provide feedback

Continuously inform the user about.

- what it is doing
- how it is interpreting the user's input
- user should always be aware of what is going on

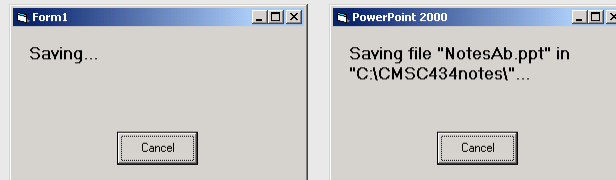


5. Provide feedback



5. Provide feedback

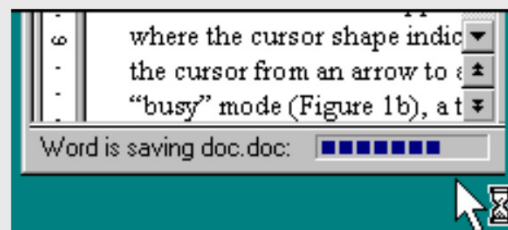
Should be as specific as possible, based on user's input.



Bad

Better

Best within the context of the action rather than with a dialog box.



5. Provide feedback

Response time is important...

- how users perceive delays
 - 0.1 second max: perceived as “instantaneous”
 - 1 seconds max: user’s flow of thought stays uninterrupted, but delay noticed
 - 10 seconds: limit for keeping user’s attention focused on the dialog
 - > 10 seconds: user will want to perform other tasks while waiting and might think that the application has failed

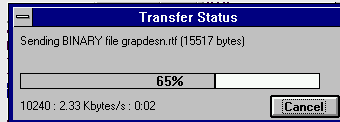
5. Provide feedback

Dealing with long delays...

- Cursors
 - for very short transactions

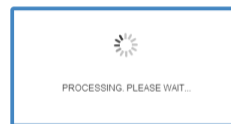
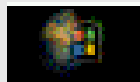


- Percent-done dialogs
 - for longer transactions
 - how much left
 - estimated time
 - what it is doing



NOTE: When giving this type of feedback, take care to do so in a meaningful fashion based upon percent of time. For example, if doing a progress bar for an e-mail client, rather than the % of messages sent, use % of size of messages.

- “Still Working”
 - for unknown/changing times



6. Provide clearly marked exits



How do I get out of this?

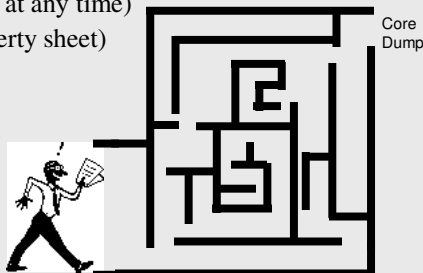
6. Provide clearly marked exits

Users don't like to feel trapped by the computer!

- should offer an easy way out of as many situations as possible

Strategies:

- Cancel button (for dialogs waiting for user input)
- Universal Undo (can get back to previous state)
- Interrupt (especially for lengthy operations)
- Quit (for leaving the program at any time)
- Defaults (for restoring a property sheet)

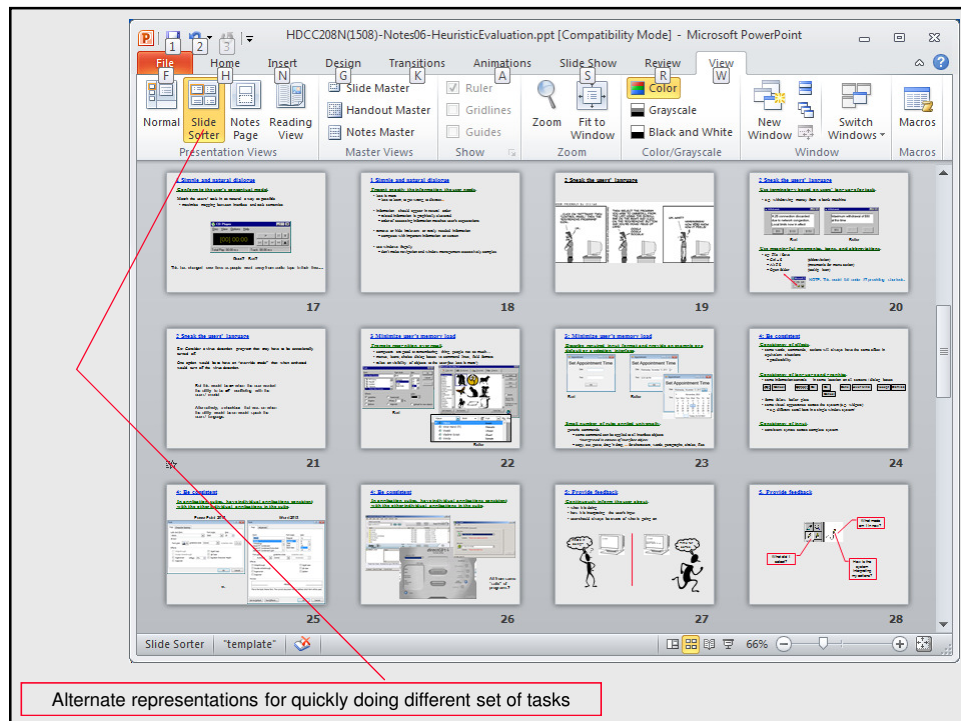
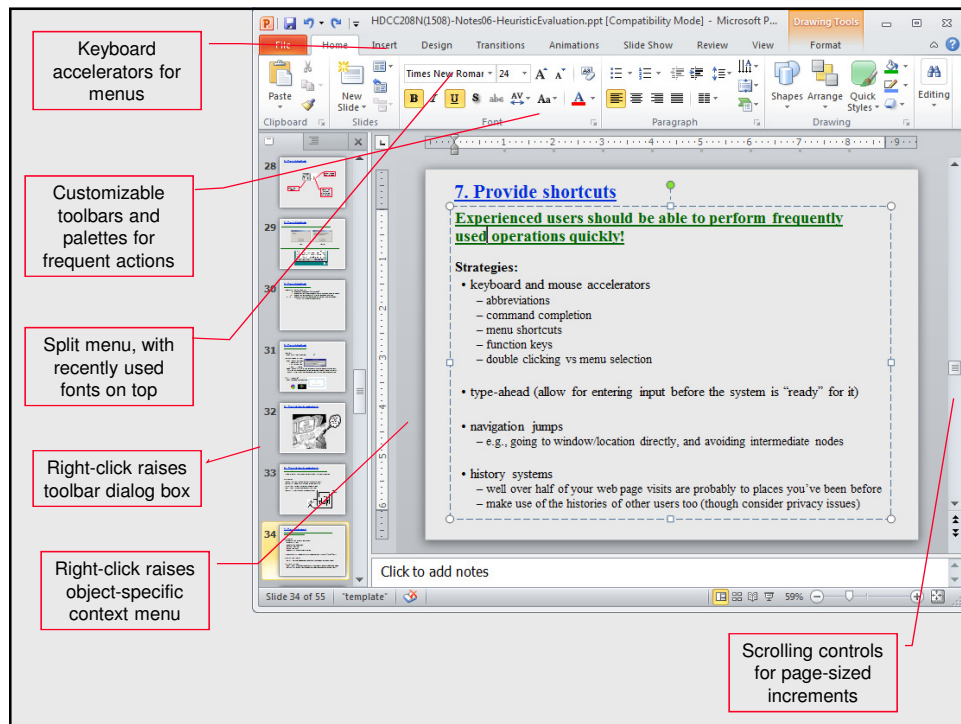


7. Provide shortcuts

Experienced users should be able to perform frequently used operations quickly!

Strategies:

- keyboard and mouse accelerators
 - abbreviations
 - command completion
 - menu shortcuts
 - function keys
 - double clicking vs menu selection
- type-ahead (allow for entering input before the system is “ready” for it)
- navigation jumps
 - e.g., going to window/location directly, and avoiding intermediate nodes
- history systems
 - well over half of your web page visits are probably to places you’ve been before
 - make use of the histories of other users too (though consider privacy issues)



8: Deal with errors in a positive and helpful manner

People will make errors – plan for it!

Errors we make

- Mistakes
 - arise from conscious deliberations that lead to an error instead of the correct solution
- Slips
 - unconscious behavior that gets misdirected en route to satisfying goal
 - e.g. drive to store, end up in the office
 - shows up frequently in skilled behavior
 - usually due to inattention
 - often arises from similarities of actions



Types of slips

Capture error (habit)

- A frequently performed activity/task can have you running “on autopilot” instead of making the choice you intended this time.
- This occurs when common and rarer actions have same initial sequence
 - William James made observations in *Psychology: The Briefer Course* in 1890 about odd triggers like going upstairs to change clothes for dinner on an occasion but starting to get ready for bed out of habit. He also commented, “Could the young but realize how soon they will become mere walking bundles of habit they would give more heed to their conduct while in the plastic state.”
 - A more modern problem is confirm saving of a file when you don’t want to replace it or saying to exit without saving when you meant to save it.

Types of slips

Description error

- intended action has much in common with others that are possible
 - usually occurs when right and wrong objects physically near each other
 - pour juice into bowl instead of glass
 - go jogging, come home, throw sweaty shirt in toilet instead of laundry basket
 - move file to trash instead of to folder

Loss of activation

- forgetting what the goal is while undergoing the sequence of actions
 - start going to room and forget why you are going there
 - navigating menus/dialogs and can't remember what you are looking for
 - but continue action to remember (or go back to beginning)!

Mode errors

- people do actions in one mode thinking they are in another
 - refer to file that's in a different directory
 - look for commands / menu options that are not relevant

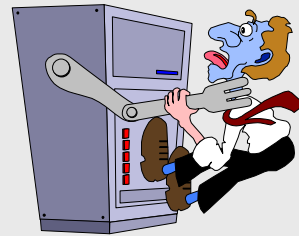
Designing for slips

General rules

- Prevent slips before they occur
- Detect and correct slips when they do occur
- User correction through feedback and undo

Examples

- capture errors
 - instead of confirmation, make actions undoable
 - allows reconsideration of action by user
 - e.g. Mac trash can can be opened and “deleted” file taken back out
- description errors
 - in icon-based interfaces, make sure icons are not too similar,
 - check for reasonable input, etc.
- loss of activation
 - if system knows goal, make it explicit
 - if not, allow person to see path taken
- mode errors
 - have as few modes as possible (preferably none)
 - make modes highly visible



Generic system responses for errors

Interlock

- deals with errors by preventing the user from continuing
 - eg cannot delete an object if none are selected

Warn

- warn people that an unusual situation is occurring
- when overused, becomes an irritant
 - e.g.,
 - audible bell
 - alert box

This page is asking you to confirm that you want to leave - data you have entered may not be saved.

Leave Page

Stay on Page

Generic system responses for errors continued...

Do nothing

- illegal action just doesn't do anything
- user must infer what happened
 - enter letter into a numeric-only field (key clicks ignored)
 - put a file icon on top of another file icon (returns it to original position)

Self-correct

- system guesses legal action and does it instead
- but leads to a problem of trust
 - spelling corrector

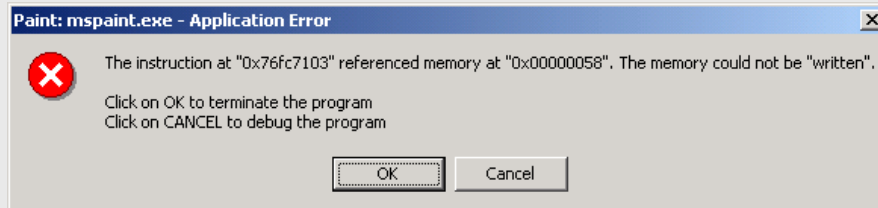
Lets talk about it

- system initiates dialog with user to come up with solution to the problem
 - compile error brings up offending line in source code

Teach me

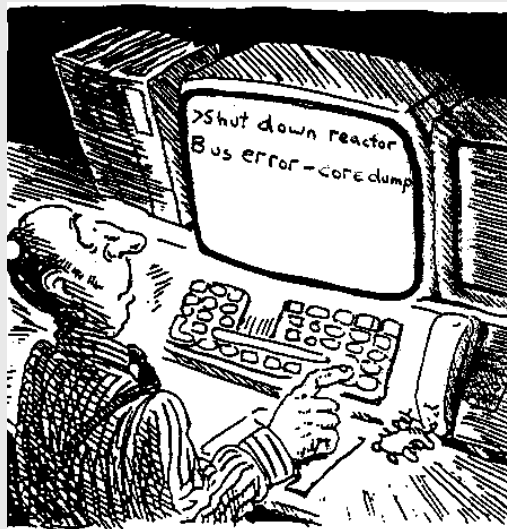
- system asks user what the action was supposed to have meant
- action then becomes a legal one

8 Deal with errors in a positive and helpful manner



HUH ??

8 Deal with errors in a positive and helpful manner



A problematic message to a nuclear power plant operator

8 Deal with errors in a positive and helpful manner

Provide meaningful error messages!

- error messages should be in the user's language (preferably task language)
- don't make people feel stupid

Bad

Try again...

Error 25

Cannot open this document.

Better

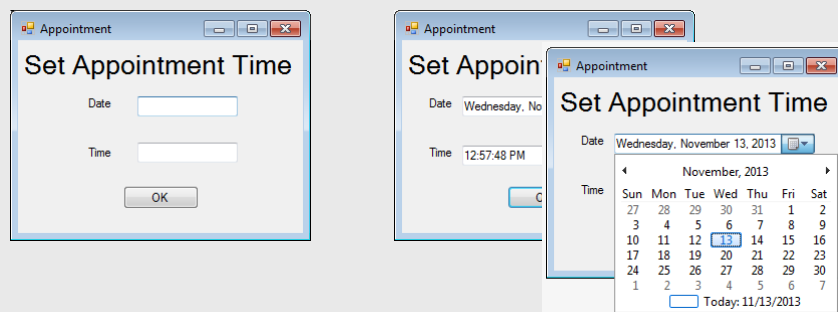
Cannot open "chapter 5" because the application "Microsoft Word" is not on your system

Cannot open "chapter 5" because the application "Microsoft Word" is not on your system. Open it with "OpenOffice" instead?

8 Deal with errors in a positive and helpful manner

Prevent errors.

- try to make errors "impossible" to make
- modern widgets: only "legal commands" selected, or "legal data" entered (which if these might allow you to enter February 29th, 2014?)

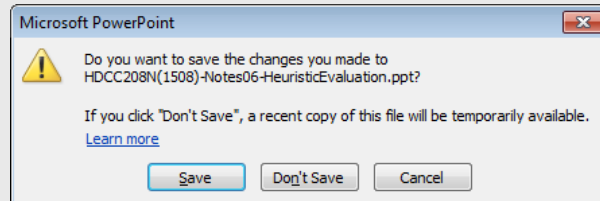


Provide reasonableness checks on input data.

- on entering order for office supplies
 - 5000 pencils is an unusually large order. Do you really want to order that many?

8 Deal with errors in a positive and helpful manner

If you know a certain type of error is common and very bad, prepare for it...



...or just avoid it completely - some applications (like Google Drive) just remove the thought of saving from the user by making it automatic (and trusting the browser won't let you close a tab/window without warning you if auto-save is pending)

Evan Golub / Ben Bederson / Saul Greenberg

Consumer Manuals...



9. Provide help

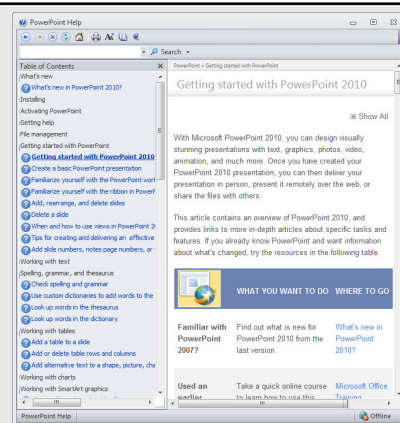
Help is not a replacement for bad design!

Simple systems:

- Should be used to walk up and use
 - minimal instructions needed

Most other systems:

- Many features available...
- Some users will want to become “experts” rather than “casual” users either right away or over time.
- Even intermediate users need reminding at times and there should be a good path toward learning things.



Documentation and how it is used

NOTE: Many users do not read manuals/documentation.

- prefer to spend their time pursuing their task

Usually used when users are in some kind of panic, they will want (and perhaps need) immediate help.

- indicates need for online documentation, good search/lookup tools
- online help can be specific to current context
- Kindle “Mayday” option? Siri/Cortana/GoogleNow option?

NOTE: paper or CD manuals are unavailable in many business environments so online/embedded help has multiple advantages

- e.g. single copy kept in system administrator’s office or even discarded

Sometimes documentation is used for quick reference in advance.

- syntax of actions, possibilities...
- list of shortcuts ...

Types of help

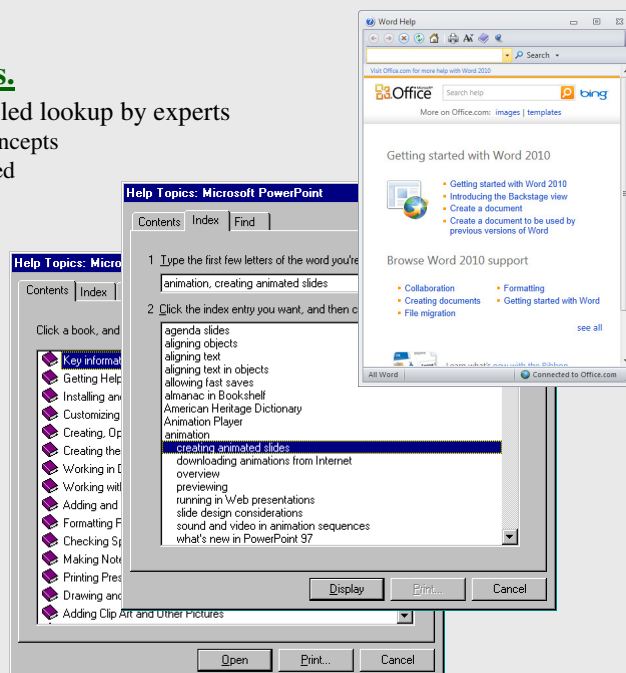
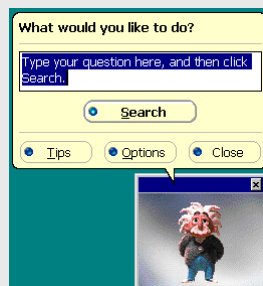
Tutorial and/or getting started manuals.

- short guides that people are likely to read when first obtaining their systems
 - encourages exploration and getting to know the system
 - tries to get conceptual material across and essential syntax
- on-line “tours”, exercises, and demos
 - demonstrates very basic principles through working examples

Types of help

Reference manuals.

- used mostly for detailed lookup by experts
 - rarely introduces concepts
 - thematically arranged
- on-line HTML
 - search / find
 - table of contents
 - index
 - cross-index



Types of help

Reminders to the user.

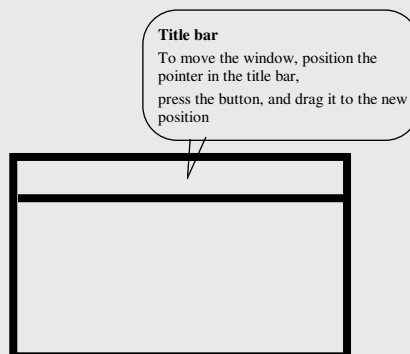
- short reference cards used to be VERY popular
 - expert user who just wants to check facts
 - novice who wants to get overview of system's capabilities
- keyboard templates used to be VERY popular
 - shortcuts/syntactic meanings of keys; recognition vs. recall; capabilities
- tooltips are STILL very popular!
 - text over graphical items indicates their meaning or purpose
 - No way to do this with touch interfaces ☹



Types of help

Context-sensitive help.

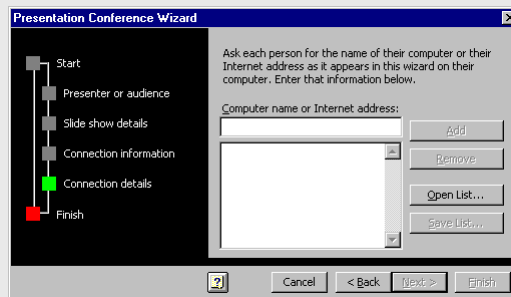
- system provides help on the interface component the user is currently working with
 - Macintosh “balloon help”
 - Microsoft “What’s this” help
 - brief help explaining whatever the user is pointing at on the screen



Types of help

Wizards specific to task.

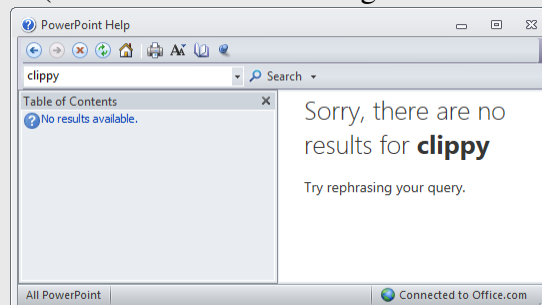
- walks the user through stages of typical tasks
- *but* very frustrating if the user gets stuck within it



Types of help

Tips to the user.

- provides migration path to learning system features
- also context-specific tips on being more efficient
- must be “smart”, otherwise boring and/or tedious and/or interrupts user’s work flow (ie: Office Assistant had good and bad elements)



Also consider: You can drag a submenu that has a move handle (a bar at the top of the menu) anywhere on the screen to create a floating toolbar.

Back Next Close



Design Principles and Usability Heuristics

- 1: Simple and natural dialogue**
- 2: Speak the users' language**
- 3: Minimize user's memory load**
- 4: Be consistent**
- 5: Provide feedback**
- 6. Provide clearly marked exits**
- 7. Provide shortcuts**
- 8: Deal with errors in a positive and helpful manner**
- 9. Provide help**