Review

- Why are Gestalt laws useful in user interface design?
- What are Gestalt laws we mentioned in the class?
- Information content in user interfaces
  - Toggle button
  - Single digit
  - Analog scale without labels
  - Audio volume
- Analog vs. digital scale

Nine Golden Rules of Interface Design

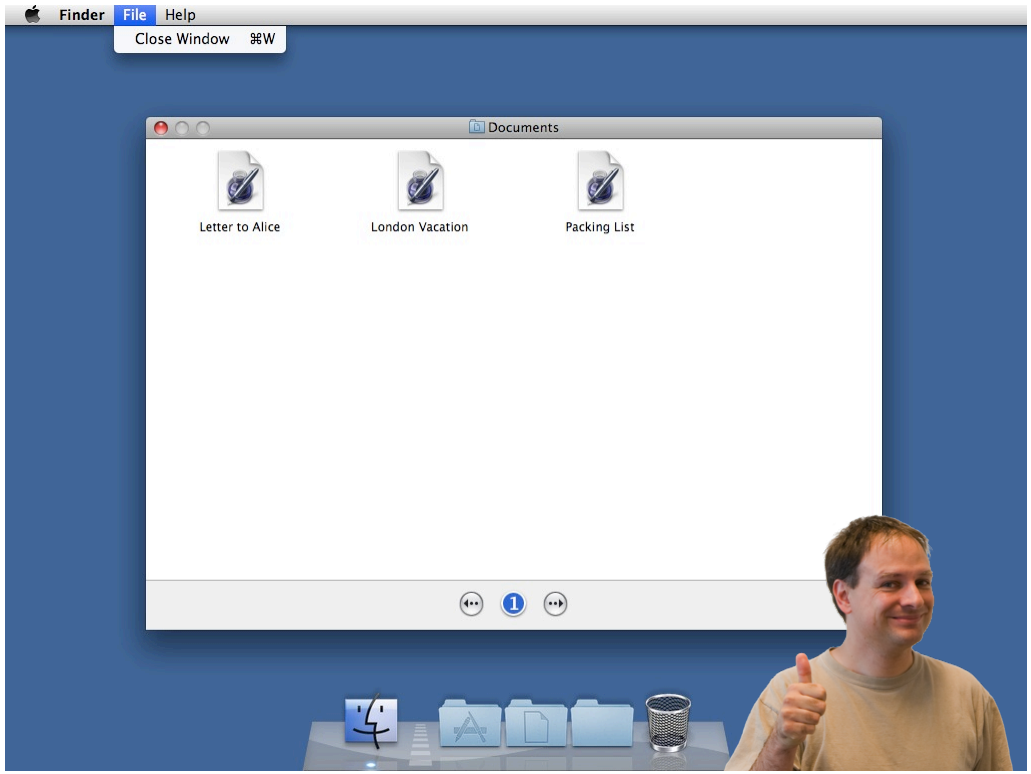
1. Keep the interface simple!
2. Speak the user's language!
3. Be consistent and predictable!
4. Provide feedback & Be responsive!
5. Minimize memory load!
6. Avoid errors, help to recover, offer Undo!
7. Design clear exits and closed dialogs!
8. Include help and documentation!
9. Address diverse user needs

1. Keep the Interface Simple!

- Most important rule
- First design often too complex & awkward
- Avoid creeping featurism
  - Others will ask for more and more features
  - But usability must not suffer
  - Experience: 80% of users use only 20% of features (e.g., Word)
  - Honorable goal would be:

    Next version will have no new features, just be easier to use
  - If pressed, move feature sets out to subdialogs

    E.g., "Simple Finder"

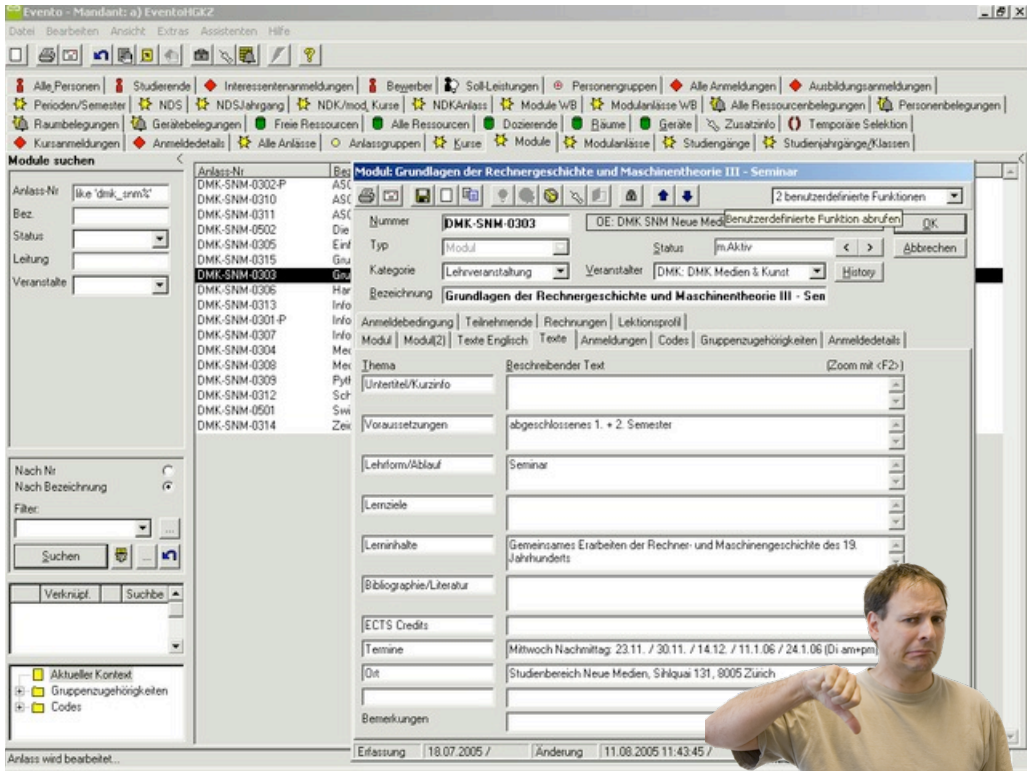


Mac OS X Simple Finder

- No double-click, no file dragging, no aliases
- 1 window, fixed size & view, no toolbars
- No folders
  - Folder creation only from within applications or through admin
- Direct access only to specified set of apps
- Problem: Applications may break simplification
  - E.g., Office ~~2008~~ 2011 settings folder

Example: VCR

Example: Simple Alarm Clock



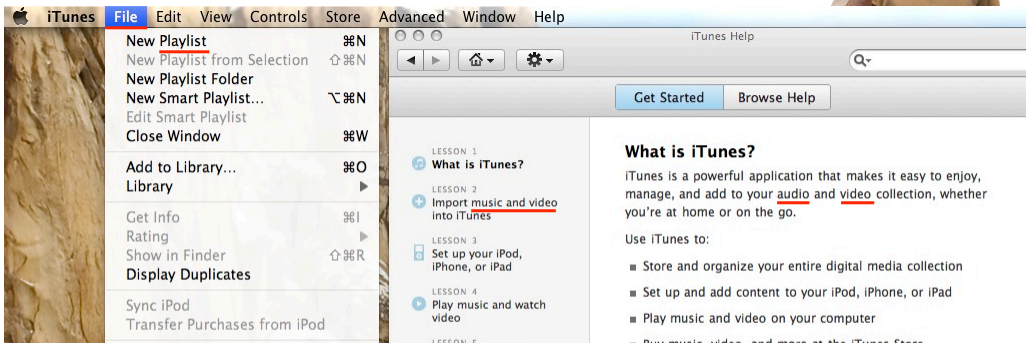
2. Speak the User's Language!

- Take words and concepts from the application domain, not computer science
- Determine terminology during initial user interviews and task analysis
- Example: "File" means less to an architect who is new to computers than "drawing"
- Applies to words for objects, but also work processes and tasks (e.g., "order")

Example: Telephone Book Menu

Example: iTunes

- Talks about "music", "songs", "video", "movies", "playlists", not "files"
  - In menus, dialogs, and online help (⇒Rule 3: Consistency)
- Exceptions: E.g., "File" menu
  - Conflict with cross-application consistency



3. Be Consistent and Predictable!

- Consistency needed on many levels:
  - Similar commands for similar situations
  - Consistent terminology in menus, dialogs, help pages
  - Consistent fonts, layout, color coding, upper/lower cases, etc. in entire system
  - Only few obvious exceptions

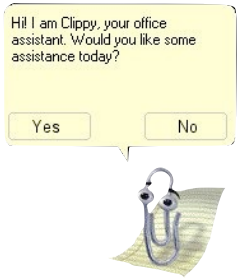
    No clear-text echo when entering passwords

    Extra security check before erasing files, etc.

Predictability

- Follow the "Principle of Least Surprise"
  - System should always react so that it minimizes the user's surprise (and therefore confusion and irritation)
- Don't do unexpected things
  - …and don't make actions unexpectedly difficult ("…how do I print this in duplex?")
- Users (especially experts) like to be "in control"
  - They initiate actions, the system responds

Principle of Least Surprise



PowerPoint Office Assistant

Time-Based Interface

Timeouts are evil!

4. Provide Feedback & Be Responsive!

- Remember the Seven Stages Of Action
  - Complete & continuous feedback bridges Gulf of Evaluation
- Each user action requires some feedback
  - Subtle for small/short/frequent actions
    - Key press, menu selection
  - More noticeable for main/long/infrequent actions
    - Saving or copying files
  - Icons in GUIs simplify visualizing object state and actions (direct manipulation)
- Nothing more frustrating than
  - "Where am I?" or "What is it doing now?!"

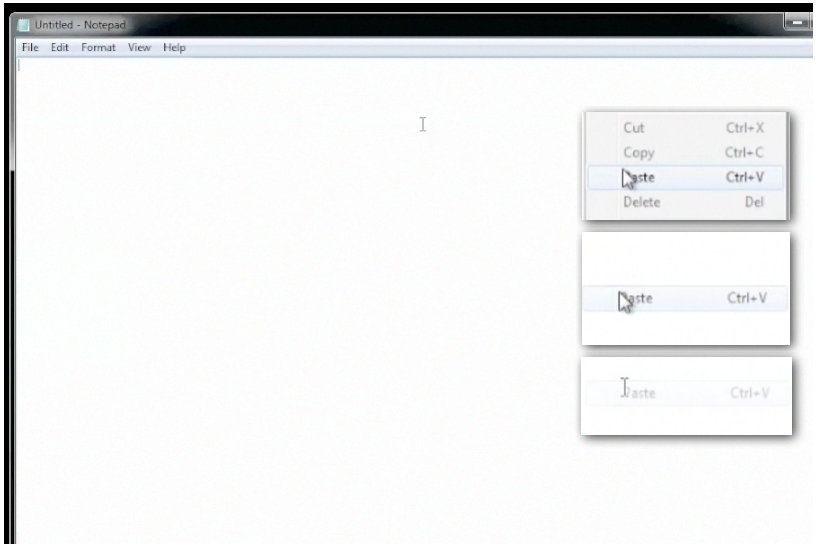
Example: Windows 2000 Progress Dialog for Copying Files

- What's wrong with this picture?

Example: Menu Selection

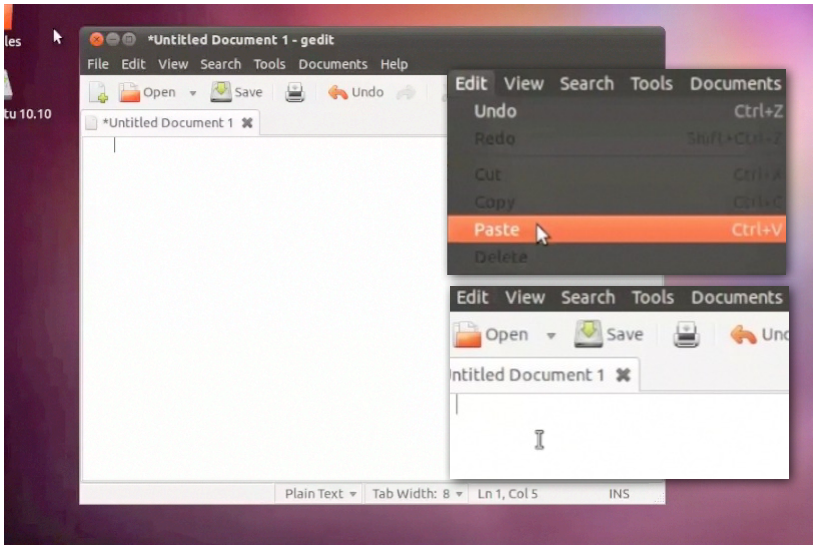
- What happens when you select a menu item?

Windows 7 Menu



Mac OS X Menu

GNOME



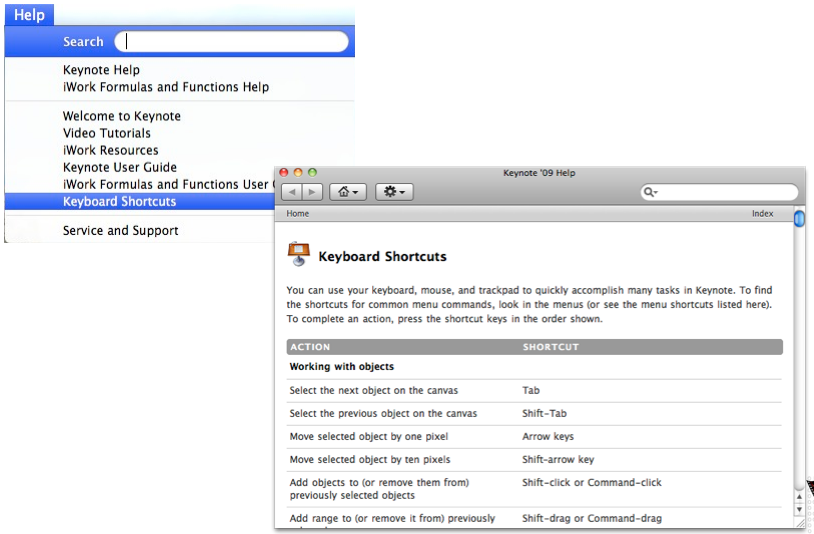
Haptic Feedback

5. Minimize Memory Load!

- Short-term memory: limited capacity
  - Ca. 7 ± 2 chunks
- Avoid situations in which prior dialog information has to be reproduced from memory
  - E.g., user should not have to type anything in twice.
- Display information so it's easy to parse
  - Gestalt laws
- Provide obvious access to help pages for codes, abbreviations, etc.
- It's easier to minimize memory load with GUIs than command line interfaces
  - "Read & Select" instead of "Remember & Type"

Keyboard Viewer

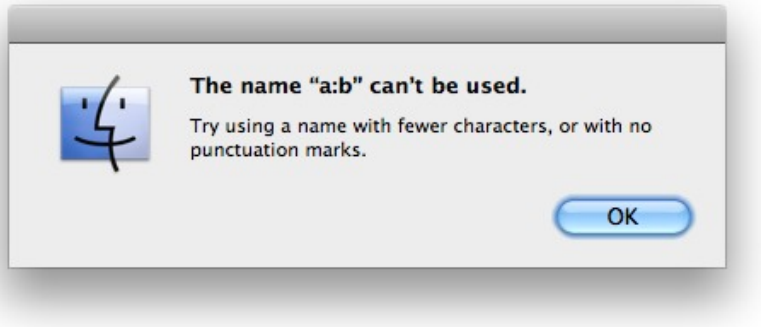
Keyboard Shortcut List



6A. Avoid Errors, Help to Recover!

- Best: Design system so mistakes cannot be made in the first place. Examples:
  - Selection instead of (mis)typing
  - Cannot type letters in numerical data fields
  - Arcade game machines have virtually no error messages!
  - Automatic correction of illegal characters in file names

    E.g., ":" in Mac OS X



6A. Avoid Errors, Help to Recover!

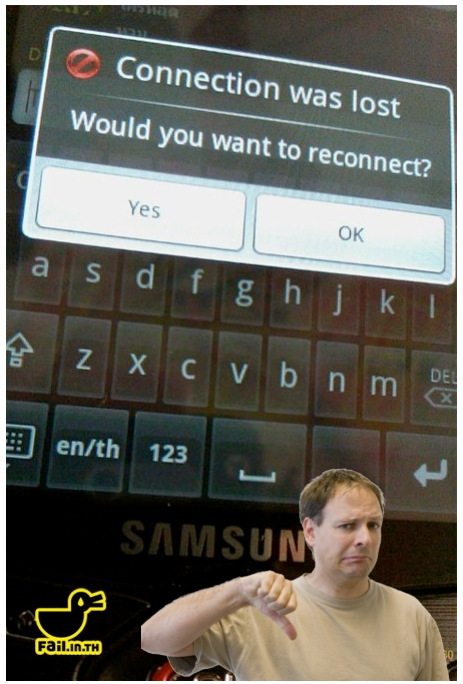
- Errors lead to stress
  - So offer simple, constructive, concrete, helpful, and comfortable instructions to recover
  - System state should not change through wrong input, or should be easy to restore

6B. Offer Undo!

- As many actions as possible should be reversible
- Lowers anxiety because users know errors are correctable
- Encourages users to try out new functions
- Ideal:
  multiple undo, and at multiple levels

7. Design Clear Exits & Closed Dialogs!

- Three most common questions of users during a dialog:
  - Where am I?
  - What can I do here?
  - How do I get back to where I was?
- Clear exits ("Back", "Quit") help with Question 3
- Closed dialogs:
  - Provide feeling of having completed a step
  - Allows user to relax, "take a breath", frees the mind for the next step



8. Include Help and Documentation!

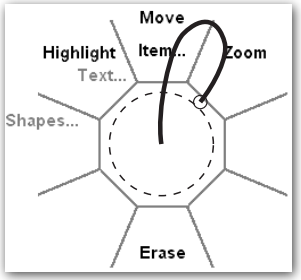
- Hierarchy of help systems, with increasing breadth and decreasing ease-of-access:
  - Dynamic Descriptors, such as Tooltips (but let users disable them!)
  - Online tutorials and references
  - Printed documentation (but:)
- More active help can be useful:
  - Assistants and Wizards
  - But danger: system takes over initiative, which breaks Rule 3 (predictability)

Users don't read manuals!

9. Address Diverse User Needs

- Novices want more explanations
- Frequent users want less fussy and faster interaction
  - They value (configurable) keyboard shortcuts, macro recording, programmability, and quick responses without unnecessary feedback (for them)
- Different age ranges have different interface expectations
- Technology affinity ("enjoying to play with gadgets") varies widely among people
- But conflict: If in doubt, Rule 1 ("Keep the interface simple") is more important! May have to focus on a user group

Example: PostBrainstorm

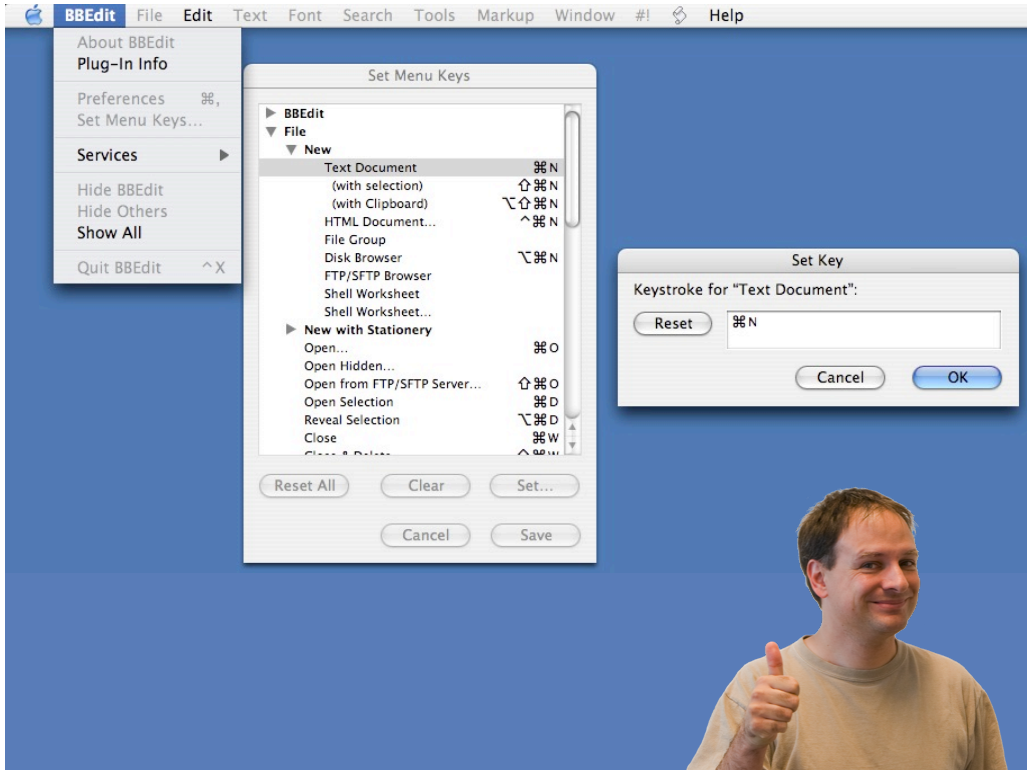


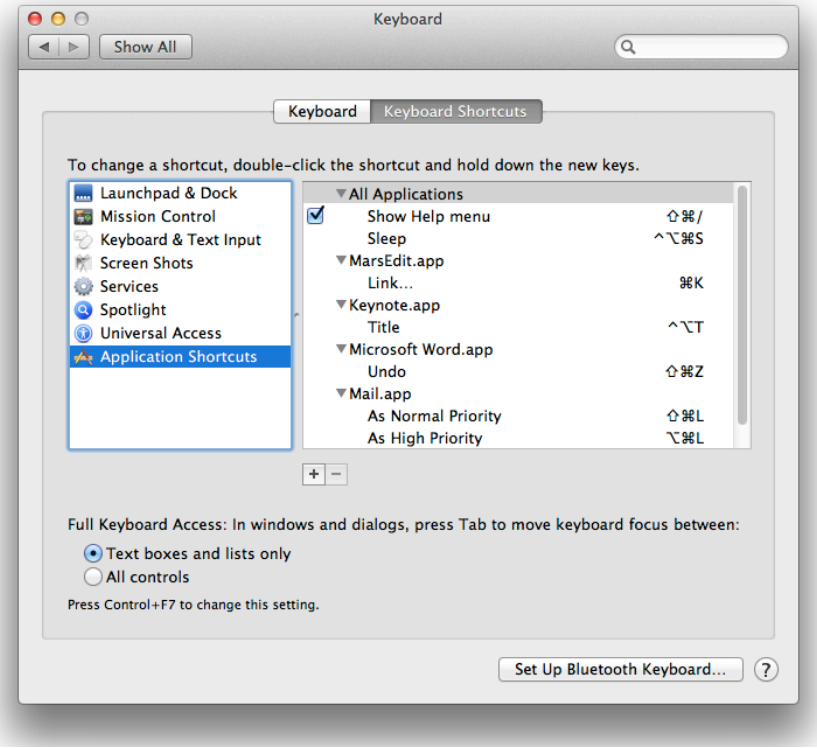
- New users get popup menu
- Experienced users remember the gestures to select frequent commands from the menu
- The menu does not even pop up when the gesture is done rapidly
- But: If you ever forget the gesture, just wait for a fraction of a second, and you can revert to using the popup menu
- The result: Fluid and reversible transition from menu selection to gesture commands

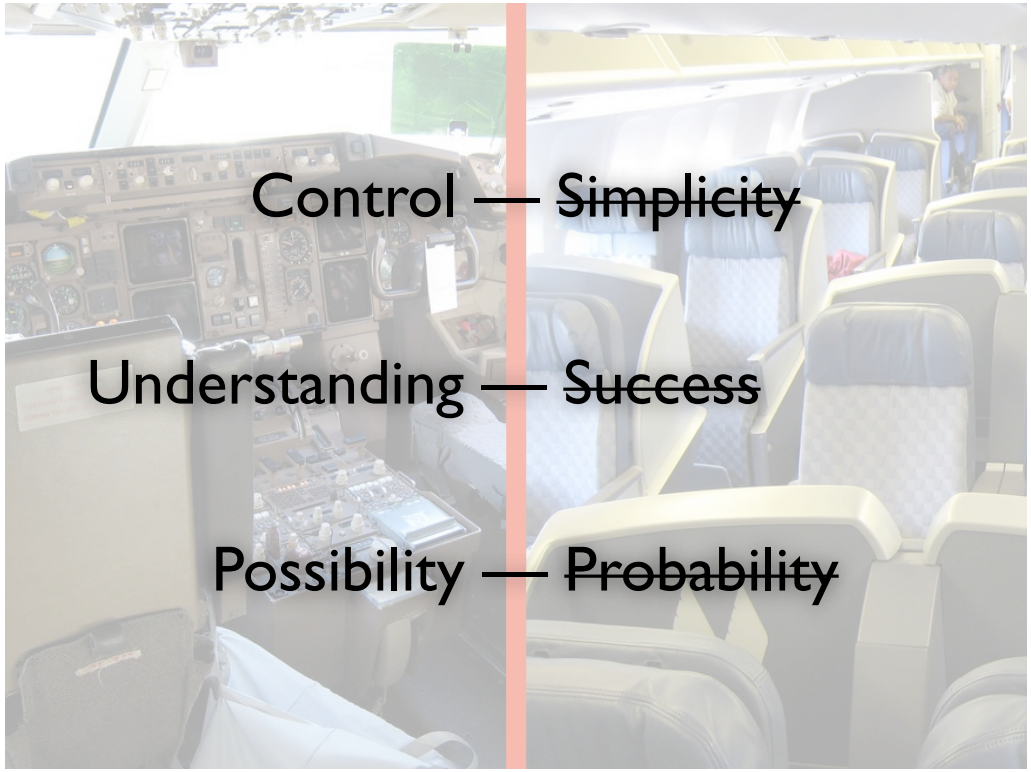
[F. Guimbretière, Stanford, UIST 2000]

Example: BBEdit

- An editor for programmers and HTML coders
- Lets users redefine most keyboard shortcuts
- Offers intuitive graphical interface to do so (since this is itself a feature most people will only use infrequently, i.e., as "selective novices")
- Includes set of Emacs-like command-key bindings for Unix people
- Compare to GNU Emacs interface to do the same thing





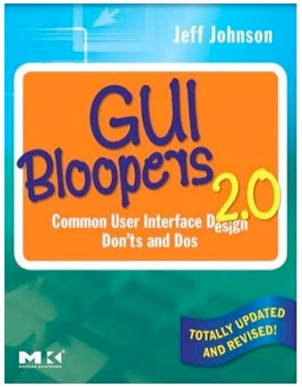


Nine Golden Rules of Interface Design

1. Keep the interface simple!
2. Speak the user's language!
3. Be consistent and predictable!
4. Provide feedback & Be responsive!
5. Minimize memory load!
6. Avoid errors, help to recover, offer Undo!
7. Design clear exits and closed dialogs!
8. Include help and documentation!
9. Address diverse user needs

Responsiveness

- See also: Jeff Johnson, *GUI Bloopers 2.0*
- Key usability problem of interactive systems
  - Bad responsiveness opens Gulf of Evaluation
- Examples for bad responsiveness:
  - A screen pointer that doesn't keep up
  - Delayed response to button-clicks
  - Sliders and scrollbars that lag
  - Applications that go "dead" during disk operations
  - Multiple screen repaints



Reasons for Poor Responsiveness

- Importance not widely known
  - UI designers think of other things first
  - UI designers rarely specify responsiveness
  - Programmers tend to equate it with performance
- This kind of tuning is always difficult
  - "We'll get it in the next release," and so on
- Developers treat human input like machine input
- Simple, naïve implementations
- GUI tools and platforms are inadequate
  - Limitations of the web (which everybody knows about)

Example: Scrollbar

- Does text move as you scroll (good) or after you let go (bad)?
- If designer doesn't specify, developer will make a decision
- That will usually be the *technically simplest*
  - Since developers are not trained in user interface theory and concepts
  - Just as UI designers are generally not trained in implementing large software products in C++

Some Eternal Facts

- Responsiveness ≠ performance!
- Processing resources will always be limited
  - We still look at hourglass as much as 15 years ago
  - UIs are real-time systems with deadlines based on human cognition
  - Software does not need to do everything instantly, or in a given order, or even at all

Three Human Deadlines

- 0.1 seconds
  - Perception of cause and effect (to be discussed after midterm)
  - E.g., delay between moving mouse and pointer following, or between mouse click and inverting button
- 1 second
  - Turn-taking in conversation, minimum reaction time for unexpected events
  - E.g., you have 1s max to show progress indicator, open window, or finish system-initiated operations (like auto-save)

Three Human Deadlines

- 10 seconds
  - Typical human attention span
  - Max. time for one step of a task

    E.g., entering a check into a banking program, or completing one step of a wizard
  - Max. time to finish input to an operation

    E.g., from selecting "Print" menu entry to sending off the print job

Design Techniques for Responsiveness

- Meet human-time deadlines
  - Rely on the three deadlines and recognize the differences
  - Acknowledge user input immediately, and display busy and progress indicators
  - Use them as frequently as you can, you never know when it will take longer
  - Example "Progress bar":

    Make it real, show total items remaining, overall progress, and estimated total time remaining

    Only useful if it advances roughly linearly! (no hanging at 99% please)

    Estimated time should always go down, never up

    "Less than a minute" is better than "47 seconds" (why?)

Design Techniques for Responsiveness

- Display important information first
  - Example: How to draw a clock
- Work in parallel
  - Delegate work that isn't time-critical to background processes
  - Work ahead by preparing likely requests
- Optimize Queueing
  - Create a logical order by looking at all pieces first, then prioritize

Design Techniques for Responsiveness

- Manage time dynamically
  - Adjust the strategy if not keeping up
  - Decrease quality or quantity to keep up
- Example: WordStar
  - Ran on a 1 MHz computer, killed by IBM PC
  - Written by an amateur, but he accommodated by making the system responsive
  - WordStar never dropped characters typed
  - Characters typed were always on screen instantly
  - Instead stopped updating other areas of the screen

Design Techniques for Responsiveness

- Test under different conditions
  - Test under heavy loads
  - Test on slower systems, like your customers have
  - Test over slower net connections

Exam Part I

- Aachen: Tuesday, 29 November 2011, 16:30–18:00
  - Aula 2 and 2010 (overflow)
- Bonn: Tuesday, 29 November 2011, 16:00–17:30
  - Main lecture hall, B-IT Center
- Cannot attend the exam?
  - Before the exam, send an email to Chat
  - Within 5 days after the exam, submit a scanned copy of the evidence
- Preparation
  - The video of this lecture last year is available on L2P
  - Post the points you need clarification on L2P discussion board before the next lab
- Aachen: There will be a lecture on 30 November