Design Principles and Usability Heuristics

You can avoid common design pitfalls by following 9 design principles

You can inspect an interface for usability problems with these principles

Design principles and usability heuristics (I)

Broad “rules of thumb” that describe features of “usable” systems

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

Heuristic evaluation
• same principles used to “evaluate” a system for usability problems
• becoming very popular
  – user involvement not required
  – catches many design flaws
• is an “expert review”
**Design principles and usability heuristics (II)**

**Advantages**
- the “minimalist” approach
  - a few general guidelines can correct for the majority of usability problems
  - easily remembered, easily applied with modest effort

- discount usability engineering
  - cheap and fast way to inspect a system
  - can be done by usability experts

**Disadvantages:**
- 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.
- subtleties involved in their use

**Discount Usability Engineering**

**Cheap/Fast/Easy To Use!**
- no special labs or equipment needed
  - might even be able to run it on your own machine in your office
  - can even be used on paper prototypes

- can be on order of 1 day to apply
  - standard usability testing may take weeks

- once understood, can use in many scenarios with little additional learning

- the more careful you are, the better it get
Heuristic Evaluation

Developed by Jakob Nielsen (1990)
• seems inspired by Shneiderman’s “Eight Golden Rules”

Helps find usability problems in a UI design

Small set (3-5) of evaluators examine UI
• independently check for compliance with usability principles (“heuristics”)
• different evaluators will find different problems
• evaluators only communicate afterwards
  – findings are then aggregated

Can perform on working UI or on sketches

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
**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

---

**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
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

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
## Nielsen’s Example Ratings List

<table>
<thead>
<tr>
<th>Rating</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>I don't agree that this is a usability problem at all.</td>
</tr>
<tr>
<td>1</td>
<td>Cosmetic problem only. Need not be fixed unless extra time is available on project.</td>
</tr>
<tr>
<td>2</td>
<td>Minor usability problem. Fixing this should be given low priority.</td>
</tr>
<tr>
<td>3</td>
<td>Major usability problem. Important to fix, so should be given high priority.</td>
</tr>
<tr>
<td>4</td>
<td>Usability catastrophe. Imperative to fix this before product can be released.</td>
</tr>
</tbody>
</table>

### 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”.

## 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
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


Why Multiple Evaluators?
Single evaluator achieves poor results
• only finds 35% of usability problems
• 5 evaluators find ~ 75% of usability problems
• why not more evaluators???? 10? 20?
  – adding evaluators costs more
  – many evaluators won’t find many more problems
Why Multiple Evaluators (cont)?

problems found

benefits / cost

(Graphs for a specific example)

1 Simple and natural dialogue

Use 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?

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
**2 Speak the users’ language**

Terminology based on users’ language for task

- e.g. withdrawing money from a bank machine

![Withdrawal images]

**Bad**

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

**Better**

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
Computers good at remembering things, people aren’t!

Promote recognition over recall
• 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 example, and default

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

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

PowerPoint 2003

Word 2003

- VS -
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

What did I select?
What mode am I in now?
How is the system interpreting my actions?

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

- how users perceive delays
  - 0.1 second max: perceived as “instantaneous”
  - 1 second 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

Dealing with long delays

- Cursors
  - for 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

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 (entering input before the system is ready for it)
- navigation jumps
  - e.g., going to window/location directly, and avoiding intermediate nodes
- history systems
  - WWW: ~60% of pages are revisits

Keyboard accelerators for menus

Customizable toolbars and palettes for frequent actions

Split menu, with recently used fonts on top

Right-click raises toolbar dialog box

Right-click raises object-specific menu

Scrolling controls for page-sized increments
8: Deal with errors in a positive and helpful manner

People will make errors!

Errors we make

- **Mistakes**
  - arise from conscious deliberations that lead to an error instead of the correct solution

- **Slips**
  - unconscious behaviour that gets misdirected en route to satisfying goal
    - e.g. drive to store, end up in the office
  - shows up frequently in skilled behaviour
    - usually due to inattention
  - often arises from similarities of actions
Types of slips

Capture error
• frequently done activity takes charge instead of one intended
  – occurs when common and rarer actions have same initial sequence
  – change clothes for dinner and find oneself in bed (William James, 1890)
  – confirm saving of a file when you don’t want to replace it

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
  – e.g. 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
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

Let’s 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**

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, bonehead!
- 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 “Teachtext” instead?
8 Deal with errors in a positive and helpful manner

Prevent errors

• try to make errors impossible
• modern widgets: only “legal commands” selected, or “legal data” entered (which if these might allow you to enter February 29th, 2011?)

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?
9. Provide help

Help is not a replacement for bad design!

Simple systems:
- walk up and use; minimal instructions

Most other systems:
- feature rich
- some users will want to become “experts” rather than “casual” users
- intermediate users need reminding, plus a learning path

Documentation and how it is used

Many users do not read manuals
- prefer to spend their time pursuing their task

Usually used when users are in some kind of panic, need immediate help
- indicates need for online documentation, good search/lookup tools
- online help can be specific to current context
- paper manuals unavailable in many businesses!
  - e.g. single copy locked away in system administrator’s office

Sometimes used for quick reference
- 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 hypertext
  - search / find
  - table of contents
  - index
  - cross-index
**Types of help**

**Reminders**

- Short reference cards
  - Expert user who just wants to check facts
  - Novice who wants to get overview of system’s capabilities

- Keyboard templates
  - Shortcuts/syntactic meanings of keys; recognition vs. recall; capabilities

- Tooltips
  - Text over graphical items indicates their meaning or purpose

**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

Title bar

To move the window, position the pointer in the title bar, press the button, and drag it to the new position.
**Types of help**

**Wizards**
- walks user through typical tasks
- *but* dangerous if user gets stuck

**What's my computer's name?**
Fred?
Intel?
AST?

**Types of help**

**Tips**
- migration path to learning system features
- also context-specific tips on being more efficient
- must be “smart”, otherwise boring and tedious