Why is Usability Important?

Why is usability important?

- We are not the users
  - Even design experts don't think the same way as all other humans
- Users are always right
  - If a user can't do what they want, the application is broken
- Decreases development costs (i.e., fewer revisions)
- Increases revenue (higher customer satisfaction => better reviews and more referrals, etc.)

What Does Usability Measure?

- Ease of learning
- Efficiency of use
- Memorability
- Error frequency and severity
- Subjective satisfaction

Usability Methods

Many methods with varying cost/benefit
Some popular approaches:

- Cognitive Walkthrough (no users)
  - Task oriented
  - Examined by experts
- Heuristic evaluation (no users)
  - Interface oriented
  - Examined by experts
- Action Analysis (no users)
  - KLM
- Focus group
- Usability test
  - Task & interface oriented
  - Run with users

Cognitive Walkthrough

Requirements:

- Description or prototype of interface
- Task description
- List of actions to complete task
- User background

What you look for:

- Will users know to perform the action?
- Will users see the control?
- Will users know the control does what they want?
- Will uses understand the feedback?

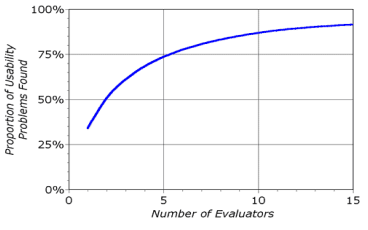
Heuristic Analysis

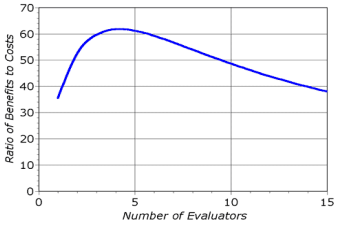
- "Rules of thumb" that describe features of usable systems
  - Can be used as design principles
  - Can be used to evaluate a design

- Pros and cons
  - Easy and inexpensive
    - *Performed by expert*
    - *No users required*
    - *Catches many design flaws*
  - More difficult than it seems
    - *Not a simple checklist*
    - *Cannot assess how well the interface will address user goals*

Usability Engineering

- Introduced by Nielsen (1994)
- Can be performed on working UI or sketches
- Requires a small set of evaluators to examine the UI
  - Check compliance with usability principles
    - *Each evaluator works independently*
    - *Go through the interface several times with different perspectives*
  - All reviews are aggregated in one final usability report





Nielsen's evaluation phases (1-2)

- Pre-evaluation training
  - Provide the evaluator with domain knowledge if needed
- Evaluation
  - First step: get a feel for flow and scope
  - Second step: focus on specific elements
    - *Multiple passes is better*
    - *Create a list of all problems*
    - *Rate severity of problem*

Nielsen's evaluation phases (3-4)

- Severity rating
  - Performed by individuals
  - Then aggregated by group
  - Establishes a ranking between problem
  - Reflects frequency, impact and persistence
    - *Cosmetic, minor, major and catastrophic*
- Debriefing
  - Discuss outcome with design team
  - Suggest potential solutions
  - Assess how hard things are to fix

Evaluation Aggregation

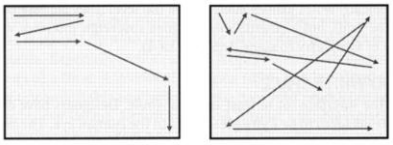
	A	B	C	D
1	**Machine 6 - Zoomable**			
2	**Severity (1-Low => 5-High)**	**Average Severity**	**Count**	**Description**
3	-,4,5	4.0	3	Color of review & cast ballot buttons should be different than progress indicator a
4	-	3.0	1	Not clear how to get started
5	?	3.0	1	Feels like a game - possibly inappropriate
6	-	3.0	1	"Not voted" confusing when multiple choices available
7	-	3.0	1	Peripheral races too visually confusing
8	1,4	2.5	2	Progress/navigation buttons is partly a progress indicator, but not clear enough
9	2	2.0	1	Overview buttons shouldn't split 4 sub-types

Nielsen's heuristics

- Simple and natural dialog
- Speak the users' language
- Minimize user memory load
- Consistency
- Feedback
- Clearly marked exits
- Shortcuts
- Prevent errors
- Good error messages
- Provide help and documentation

Simple and natural dialog

- Present information in natural order

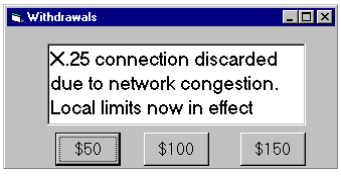


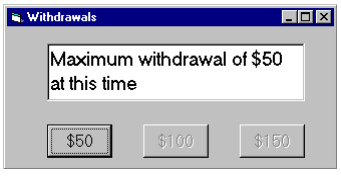
From Cooper's "About face 2.0"

- Simple is good
  - Remove or hide irrelevant or rarely needed information
    - *They compete with important information on screen*
      - Pro: Palm Pilot
      - Against: Dynamic menus
  - Use windows frugally
    - *Avoid complex window management*

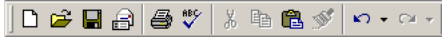
Speak the users' language

- Use a language compatible with users' conceptual model
  - Example: withdrawing money at an ATM



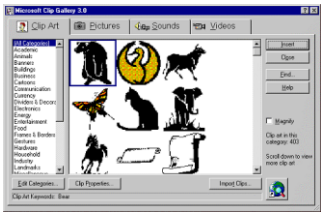


- Use meaningful mnemonics, icons and abbreviations

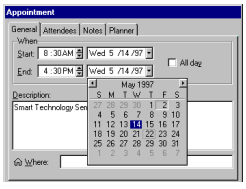


Minimize user memory load

- Promote recognition over recall
  - Recognition is easier than recall



- Describe expected input clearly
  - Don't allow for incorrect input



- Create orthogonal command systems
  - Using generic commands that can be applied to all interface objects

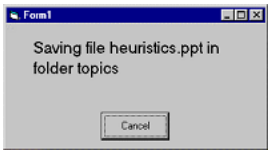
Consistency

- Be consistent in
  - Command design
    - *Same action, same effect in equivalent situations*
  - Graphic design
    - *Input format*
    - *Output format*
  - Flow design
    - *Similar tasks are handled in similar ways*

- Consistency promotes skills acquisition and/or transfer

Feedback (Semantic)

- Users should always be aware of what is going on
  - So that they can make informed decision
    - *Be specific*

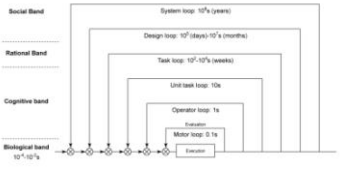


  - But do not overburden users!
  - Provide redundant information

Feedback: Toolbar, cursor, ink

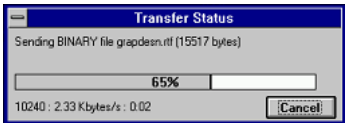
Feedback (Time)

- Different feedback time scales
  - Shall I wait for that task to finish or go for coffee?



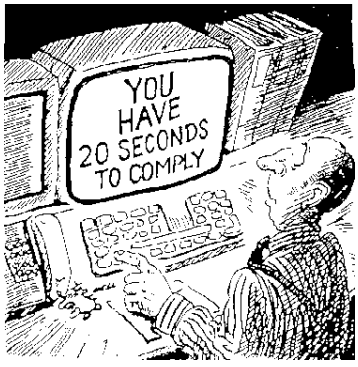
> 10s	User will switch to another task while waiting
10s	Difficult to stay focused
1s	Delay but user's flow of thought is uninterrupted
.1s	Causality

- Different techniques
  - Short transaction: hour glass cursor
  - Longer transaction: estimate of time left
    - *An overestimate is always better!*



Clearly marked exits

- Users don't like to be trapped!



- Strategies
  - Cancel button (or Esc key) for dialog
    - *Make the cancel button responsive!*
  - Universal undo

Shortcuts

- Expert users should be able to perform operations rapidly
  - Try to limit the training necessary to access advanced features
- Strategies
  - Keyboard and mouse accelerators
    - *menu shortcuts and function keys*
    - *command completion, command abbreviations and type-ahead*
  - Toolbars and tool palettes
    - *Trade screen real estate for rapid access*
  - Navigation jumps
    - *History systems*
      - 60% pages are revisits

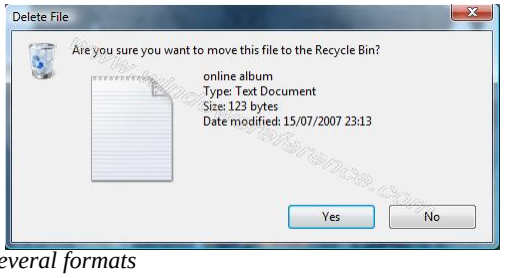
Preventing errors

- Be tolerant
- Error types
  - Mistakes
    - *Conscious decision with unforeseen consequences*
  - Slips
    - *Automatic behaviors kicking in*
      - Drive to the store, end-up in the office
      - Press enter one time too many…
    - *Mode errors*
      - Forget the mode the application is in
    - *Loss of activation*
      - Forget what your goals were

Designing for slips

An ounce of prevention is worth more than a pound of cure!

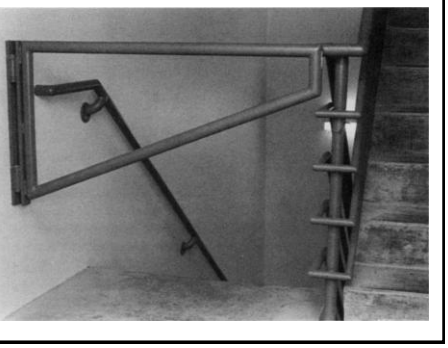
- Examples
  - Design modeless interfaces
  - Instead of confirmations provide undo mechanisms



  - Check for reasonable input
    - *Be prepared to handle several formats*
    - *Make entering a incorrect format impossible*
  - Make the current goal clear
    - *Prevent lost of activations*

Forcing functions

- Interlock mechanisms
  - Require step A before step B can be performed
  - Ex: Switching from P to D in a car requires pressing brake pedal
- Lockin mechanisms
  - Process continues unless user removes constraint before stopping it
  - Ex: No eject button for floppy disk on Mac
- Lockout mechanisms
  - Process won't occur unless user removes constraint before starting it
  - Ex: Basement stairway



Heuristic Analysis Exercise

www.kayak.com

Focus on flight search

1. Simple and natural dialog
2. Speak the users' language
3. Minimize user memory load
4. Consistency
5. Feedback
6. Clearly marked exits
7. Shortcuts
8. Prevent errors
9. Good error messages
10. Provide help and documentation

Usability Study - Qualitative approach

- Gather user's perception of the interaction
- Concerned more about *ability* to use system than how much they like it

- Methods
  - Direct observation
    - *Simple observation*
    - *Thinking aloud*
    - *Constructive interaction (co-discovery)*
  - Interviews, questionnaires and surveys

Direct observation

- Observing (and recording) users interacting with the system
  - In lab or in the field
  - For a set of pre-determined tasks or through normal duties
    - *Be prepared!*

- Excellent at identifying gross design/interface problems

- Three general approaches:
  - simple observation
  - think-aloud
  - constructive interaction

Be prepared!

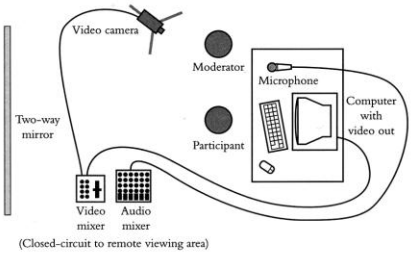
- Select the correct population
- Set objectives and tasks
  - Realistic
  - Informative
- Hardware
  - Computer, video equipment…
- Software
  - Up and running, properly debugged…
- Facilitator
  - Using a checklist might be useful
  - Practice!

Creating tasks

- Describe in terms of end goals
- Specific and realistic
- Doable
- Not too long (< 5-10 minutes each), or shorter

Recording observations

- Need a record
  - Further analysis
  - Proof during discussion
- Techniques
  - Paper and pencil
    - *Simple to set up*
      - Free form
      - Coding scheme
    - *Might be biased*
  - Audio/Video recording
    - *More accurate*
    - *Time consuming to analysis*
      - Encoding is a time consuming process



From "Observing the user experience" (Kuniavsky)

Coding scheme example

- Tracking activity in the office

s	**start**
e	**end**

	Desktop activities			**Absences**		**Interruptions**	
Time	Computer	Desk	Telephone	Desk	Room	Visitor	Telephone
9:00	s						
9:02	e					s	
9:05					s	e	
9:10			s		e		
9:13							

Simple observation method

- Evaluator observes users interacting
  - Sometime behind a half-silvered mirror
- Drawback
  - No insight into the user decision process or attitude

The think aloud method

- Subjects are asked to say what they are thinking/doing
  - What they believe is happening
  - What they are trying to do
  - Why they took an action
- Widely used in industry
- Drawbacks
  - Awkward/uncomfortable for subject (thinking aloud is not normal!)
  - "Thinking" about it may alter the way people perform their task
  - Hard to talk when they are concentrating on problem

Facilitator's Role

- Support participant
- Support designers/developers
- Accurately collect data

The constructive interaction method

- Two people work together on a task
  - Normal conversation between the two users is monitored
    - *removes awkwardness of think-aloud*
  - Variant: Co-discovery learning
    - *Use semi-knowledgeable "coach" and naive subject together*
    - *Make naive subject use the interface*
- Drawback
  - Need a good team

Debriefing

- Post-observation interviews
  - Questions from your notes
  - Questions from users diary
  - Questions from a video footage
- Pros and Cons
  - Avoids erroneous reconstruction
    - *Provide many constructive suggestions*
  - Time consuming
    - *But extremely valuable*

Interviews

- Method
  - Pick the right population
    - *Individual or group discussion*
  - Be prepared
    - *Plan a set of central questions*
  - Probe more deeply on interesting issues as they arise
    - *Focus on goals not technology*
    - *Find the root of the problem*
- Pros and cons
  - Very good at directing next design phase
    - *Provide many constructive suggestions*
  - Subjective
    - *Do not ask leading questions*
  - Time consuming

Questionnaires and surveys I

- Method
  - Pick the population
    - *Demographics and sample size*
      - Between 50 and 1000 subject
  - Establish the purpose of the questionnaire
    - *What information is sought?*
    - *How would you analyze the results?*
  - Establish the means of delivery/collection
    - *On-line*
    - *Direct interaction with users*
      - Walking in the street
      - Post-user testing
    - *Surface mail*
      - including a pre-addressed reply envelope gives far better response

Questionnaires and surveys II

- Method
  - Design the questionnaire
    - *Don't forget to debug it!*
  - Deliver
  - Collect and analyze the data
  - Establish the main findings

Closed questions

- Supply possible answers

  Characters on the computer screen are:

  hard to read easy to read

  1 2 3 4 5

  – Easy to analyze
  – Make it more difficult for respondents

Style of closed question: Scalar

– Likert Scale

  Characters on the computer screen are:

  hard to read easy to read

  1 2 3 4 5

– Be sure to pick odd numbers of choice
  - *Often 5 or 7*

Style of closed question: Multi-choice

Which types of software have you used? (tick all that apply)

[ ] word processor
[ ] data base
[ ] spreadsheet
[ ] compiler

– Can be exclusive or inclusive

– Be sure to be specific

Do you use computers at work:

[ ] often [ ] sometimes [ ] rarely

vs

Do you use computers at work:

[ ] more than 4 hrs [ ] between 1 and 4 hrs [ ] less that 1 hrs

Style of closed question: Ranked choice

Rank the usefulness of these methods of issuing a command

(1 most useful, 2 next most useful..., 0 if not used)

__2__ command line
__1__ menu selection
__3__ control key accelerator

– Helpful to understand users preference

Rating vs. Ranking?

Open ended questions

- The user answers in his/her own words

  Can you suggest any improvements to the interfaces?

  - Good for general information
  - Difficult to analyze
    - *Need for a coder*
  - Can complement closed questions

Questionnaires and surveys

- Pros and cons
  - Preparation is expensive
    - *Need to design and debug the questionnaire*
  - Can reach a large population
    - *But often a low return rate*
    - *Sample not necessarily representative*
  - As good as the questions asked
  - Data collection can be tedious
    - *Use automatic forms for large volume*
    - *Open ended questions hard to analyze*

Resources

- www.usability.gov
- www.utexas.edu/learn/usability/index.html
- www.useit.com/alertbox/mobile-usability.html

- *Don't Make Me Think* by Steve Krug
- *Usability Engineering* by Jakob Nielsen
  (the classic – 1993!)

Usability Study Exercise

- Using only www.cs.umd.edu/hcil
  - Determine what an undergrad can do with the HCIL
  - Which professors are doing something you might be interested in?
  - Is anyone doing research related to how people program?