The UNIVERSITY of VERMONT

Benjamin Fenster
CS-296 (Damon)
15 March 2005

Evaluation

I. Introduction
   a. Remember the phases: Understand, Design, Implement, Evaluate.
   b. Goals
      i. Easy to Learn
      ii. Efficient to Use
      iii. Aesthetically Pleasing
      iv. There are different techniques for evaluating easy.
   c. Forms of Evaluation
      i. Use it yourself
      ii. Let a colleague use it.
      iii. User testing.
      iv. Heuristic Evaluation
      v. GOMS

II. Use it Yourself
   a. Tells you nothing about ease of learning.
   b. This is nothing but a quick first test.

III. Let a Colleague Use It
   a. Colleagues know the technical details, but not how it actually works.
   b. Never publish something or show it to a user if you're embarrassed about showing it to a co-worker. It just makes no sense.

IV. Heuristic Evaluation
   a. Focused on ease of learning.
   b. Very subjective.
   c. More experienced evaluators find more and more issues. More developers working together find more problems.
   d. Guidelines
      i. Simple and Natural Dialog. Interface is arranged so what you want to do next is obvious (but don't clutter with extraneous options)
      ii. Graphic Design and Color
         1. Want consistency!
         2. Colors and fonts have meanings
         3. Use something visual to highlight groups of controls. Support chunking! Also keeps the user focused on the right things.
      iii. Less is More.
      iv. Speak the User's Language
         1. Write from the user's perspective ("You have bought" vs. "We have sold you.")
         2. Use consistent terminology
         3. The metaphor should make sense. A paper shredder implies the document has been destroyed. Don't use that in place of a Recycle Bin.
      v. Consistency (to language, visual, *steps*)
      vi. Feedback. Never leave the user guessing about what the computer is doing.
      vii. Clearly Marked Exists
         1. Always give a way out.
         2. When backing out, take back to the most reasonable place (don't just go back to the very beginning of the whole sequence of steps if you can back up just one step and leave the user where s/he was)
      viii. Shortcuts. Keep the option for "the hard way" but allow experienced users a better option.
      ix. Good Errors.
         1. What data was bad? How can it be fixed?
         2. Always tell the user how to fix it.
         3. Allow easy recovery from the error.

x. Prevent Errors
    1. Avoid modes. That means things you already know how to do may or may not work (if you're in the wrong mode)
    2. Tell (via example) what data should look like
xi. Help and Documentation
    1. Few people will read the docs in advance
    2. Online, context sensitive help is *good*
    3. When something's disabled, give a clue about how to enable it (for the cut button, say "select some text to cut it.")

V. Prototyping
a. Goals – Dix Perspective
    i. Learnability
        1. Predictability
        2. Synthesizability – Ease of modeling what's really happening
    ii. Flexibility
        1. Who's in control of the dialog
        2. Multi-threading: says responsive while working
        3. Task migratability: ease of transferring control to the computer
    iii. Robustness
b. Prototypes
    i. Very common in GUI design
    ii. Want to see the interface in time to make changes.
    iii. Vary by Fidelity
        1. High Fidelity
            a. Very close to how the final product will work.
            b. Full UI with limited functionality.
            c. RAD tools (VB, HyperCard) make it faster to build, then throw away when you're done.
        2. Low Fidelity
            a. Just gives a sense of how it works.
            b. Looks roughly like the final product, but not at all polished.
            c. Sketches on paper are good.
            d. Generates dramatically different comments from testers than a High Fidelity prototype would
        3. Use Low-Fi if you want comments on *flow*.
        4. There are tools to automate low fidelity prototypes
        5. Most commonly: Use sheets of paper and have someone (Wizard of Oz) switch them around for the user.
        6. Kindergarten Stuff
            a. Scissors, glue, hand drawing
            b. Screenshots from other sources
            c. Takes three to four hours to get a good sized prototype
            d. Separate pieces of paper for objects that drag (can drag with finger)
            e. Can feverishly draw new screens if the user does something totally unexpected
    iv. Advantages
        1. Don't have to learn new skills
        2. Encourages group interaction
        3. Easy to make changes (both during development and during evaluation)
        4. If the user thinks something is a button, make it a button! Can't do that with a real application, but can easily do it with the computer.
        5. User focuses on flow, not on fonts and icon positioning.
        6. Can keep the user engaged for longer by making changes – don't have to stop and explain how it works.
    v. Vary by Underlying Support: Database, constants, random number generator?

vi. Goals change over time. Toward the end you're very concerned with things like icon placement. At the beginning you're not.
vii. Long-Term Intent
1. Throw-Away: Your *intent* is to trash the whole prototype.
2. Draft: Keep some of it, but incur some risk in having old code.

c. Denim / Silk
i. Done as academic projects, and they look like it.
ii. Really require a stylus.
iii. It's not as easy to make changes on the fly as with real paper.
iv. With a tablet PC, can go to a meeting and sketch what they're describing, then run it on the spot. That's impressive.

d. Draw Packages (e.g. Visio) give a crisp look, but there's no run mode.

e. Visual Basic
i. Can add behaviors that may be hard to describe any other way
ii. At this point, maybe you should be writing the real code

f. MacroMedia Director
i. Built on the movie paradigm (cast, stage, score)
ii. Animation, interaction, …
iii. With the player, can run on most platforms.
iv. Has replaced HyperCard, largely.
v. Can get a free trial download.

VI. Gentleness
a. A word of caution. People get passionate about interfaces.
b. Be gentle when critiquing.
c. Need to work with the designer, not against.

VII. GOMS
a. Goals, Operators, Methods, Selections
b. Evaluates efficiency objectively
c. Works really well
d. Case Study
i. Nynex. Employs tons of operators.
ii. Costs about $1,000,000 per second per call per year.
iii. Someone built a GUI to improve efficiency, but it actually hurt!
iv. Hired Bonnie John as a consultant; found which areas were responsible for the slow-down via GOMS.
v. Correctly predicted that the GUI was 10% worse using GOMS and suggested improvements to boost 25%.

e. Goals
i. Task you want to accomplish.
ii. Fairly low-level. "Delete a word."

f. Operators
i. Low-level actions
ii. Press a key, move hand to mouse.
iii. Can build a hierarchy in some models where goals at one level become operators at the next.

g. Methods. Sequence of operators to achieve a goal.
h. Selection Rules. How to pick methods.
i. Using
i. Remember, all empirically derived
ii. New input devices have to be analyzed before you can predict anything.
iii. Stated goal: 80% accuracy for 20% effort
iv. Can't be too precise due to variations
v. Also assume users are experts

j. Variants
i. Key Level Model (KLM)
1. Always low-level

2. Cognitive isn't a big part

ii. CNM GOMS

iii. NGOMSL
   1. Richer
   2. Includes time for learning simple things

iv. CPM GOMS
   1. "Cognitive Perceptual Motor"
   2. Also: "Critical Path Method"
   3. Because: we're doing things in parallel.
   4. Doesn't apply everywhere, but it's very accurate where it does apply.

k. KLM GOMS

i. One thing at a time.

ii. No learning.

iii. Operators.

K press key	P point	H move hands to / from mouse
D draw line segment	M mental activity	R response time (of computer)

iv. Very simple, but troublesome to list all the steps correctly.

l. CPM GOMS

i. Based most directly on the model human processor

ii. Allows parallelism (cognitive, motor, verification)

iii. New operation V verification (did I click the right spot?)

iv. Very low level. Move eyes, perceive location of text, et cetera.

v. Overlapping
   1. See [CS-296-2005-01-SLIDES-10:28]
   2. Can move eyes while initiating cursor movement.
   3. Realistically models how things are happening mentally.

vi. Clearly very painful to use, but also very accurate

vii. Gets slightly *faster* times, which is more accurate for people who are doing the same things over and over.

viii. Decision-making isn't as much a part of this as KLM.

m. NGOMSL

i. Incorporates learning into the model.

ii. Cognitive Complexity Theory

iii. Uses program form
   1. Steps (goals)
   2. Explicit return
   3. "Method for goal : cut text"

iv. Selection Rules. How to accomplish a goal depends on context. Costs no time to make a selection!

v. Memory
   1. Includes storing and retrieving from short- and long-term memory
   2. Short-term requires zero time.

vi. Results
   1. Times tend to be longer.
   2. Includes time for verification.

vii. Learning
   1. Takes about 17 seconds to "learn" one step.
   2. It takes 13 minutes to learn "cut and paste." That means that if you figure out the fastest you'll ever do it, then add up all the extra time you spend cutting and pasting before you get to that point, it'll be 13 minutes.

n. GLEAN3

i. GOMS models are hard to build – tedious and error-prone.GLEAN3 is a tool to automate the process.

ii. Build GOMSL steps, then "run" them in GLEAN to see if they actually work to accomplish the task.

iii. First have to define all methods, goals, mental state of the user (what she already knows).
iv. Also need a finite state machine of the program and how it will react.

o. Wrap-Up
i. Describes how fast the user is, not how fast the interface is.
ii. Good: Can use during development without even having to build a prototype.

VIII. User Testing

a. Counts as human subject testing in universities, and is subject to the same paperwork / process as medial research, et cetera.

b. Goals
i. How clear is the interface? Compliments GOMS.
ii. What does the user want? Focus groups are better, but user testing is okay for that.

c. Setup
i. Moderate sized room (bigger than UVM's graduate offices)
ii. Two users
iii. One facilitator who can give hints when the user gets really frustrated
1. Has to be someone who won't look disappointed! Users will feel you're disappointed with them, not with the interface.
2. Introduce the product (possibly up to hours of training or as little as a quick interview).
3. Also find out about the users' backgrounds
iv. Recording
1. Don't want users to see you're doing it.
2. Record everything they're doing.
3. May have many live observers, but keep them out of the room. Otherwise they'll groan and sigh.
4. Remote video (even draping a wire across the hall through two shut doors is fine).
5. Fancy facilities will have soundproof one-way glass.

d. Task Selection
i. Very important!
ii. Task should be about an hour in length
iii. If efficiency testing, may include additional training (maybe give them the software to use at home for a while, just don't give them the actual task you'll use in testing for timing)

e. User Selection
i. Get people like your users
ii. Expect to pay (at *least* free lunch and expenses like parking, or a copy of the product)
iii. Do *not* use friends and family. They'll be unrealistically determined to please.
iv. Do *not* use new employees. They'll feel too much pressure.
v. Where to find users
1. Want ads, existing customers (but that's risky)
2. Easy to get teens for a few bucks.
vi. Use pairs so you can hear conversation (why did they do something the way they did), but don't use pairs who know each other already (or you may not understand the conversation – "it was like that other thing!")

f. Followup
i. Make changes! Otherwise why bother?
ii. Have to know what confused the user. Then add clarification (more on that later).