Lecture 7: Prototyping & Evaluation

Ideas to windows

- How do we go from ideas to windows?
  **Prototyping...**
  - rapid initial development, sketching & testing
    many designs to determine the best (few?) to continue w/ further design & implementation

- Why do we prototype?
  - get feedback on our design faster
  - experiment with alternative designs
  - fix problems before code is written
    • --> saves time! saves money!
  - keep the design centered on the user

Progressive refinement

- Applications are iteratively refined
  - from less to more detail
  - from coarse to fine granularity

- Designers create representations of app, web sites, etc. at multiple levels of detail

Levels of prototyping

- **Fidelity** refers to the level of detail
- **High fidelity**
  - prototypes look like the final product
- **Low fidelity**
  - artists renditions with many details missing

Thanks to James Landay @ UC Berkeley for the foundation of these slides!
Hi-fi prototyping (➔ implementation)

• IDE = Integrated Development Environment
  - provides editor, compiler, debugger, etc.
  - also provides “builder” for GUIs

• Example: Borland JBuilder
  - creating a new ...

Hi-fi prototyping (➔ implementation)

- specifying the main application frame
  - when you click on “Finish”, what does this do?

Hi-fi prototyping (➔ implementation)

- creation of the frame class(es) / file(s)

Hi-fi prototyping (➔ implementation)

- refining the design
  (for a JButton)
Hi-fi prototyping (→ implementation)

- And continuing with implementation...
  - compiling and running the application
  - debugging
  - final production of the application
    (or applet, servlet, ...)

Issues with hi-fi prototyping

- Advantages
  - don’t need to produce layouts from scratch
  - initializes all the basic component code,
    you fill in the functionality later

- Disadvantages
  - must give specific instance of a general idea
    - e.g., exact widgets, fonts, alignments, colors
  - designers, evaluators focus on details
    instead of overall structure & functionality
  - takes longer!

Why use low-fidelity prototyping?

- Traditional methods take too long
  - sketches → prototype → evaluate → iterate
- Can simulate the prototype
  - sketches → evaluate → iterate
  - sketches act as prototypes
    - designer “plays computer”
    - other design team members observe & record
- Kindergarten implementation skills (?!)
  - allows non-programmers to participate

Paper sketches

Course manager

Music player
Storyboards

- What are “storyboards”?
  - used in film & animation
  - provide a “script” of important events
    - leave out the details, concentrate on key interactions

“Luke & Leia coming toward camera. Behind them, Biker #3...”


Storyboards for designing GUIs
- create / pin-up lots of screens
- specify user interaction by associating screens

Low-fidelity prototyping

- Advantages of low-fi/paper prototyping
  - support brainstorming
  - do not require specification of details
  - designers feel comfortable sketching

- Drawbacks
  - do not evolve easily
  - lack support for “design memory”
  - force manual translation to electronic format
  - do not allow end-user interaction

- Can we do better?

Research Question of the Day

- Quickly sketch this...
Research Question of the Day

• Add behavior...

Research Question of the Day

• Transform it to this...

Research Question of the Day

• SILK = Sketching Interfaces Like Krazy!
  • Designer sketches ideas rapidly with electronic pad and pen
    - SILK recognizes widgets
    - easy editing with gestures
  • Designer or end-user tests interface
    - widgets behave
    - specify additional behavior visually
  • Automatically transforms to a “finished” GUI

Research Question of the Day

• SILK: Screens
  - screen = rough sketch of single screen state, including layout & components
  - features for handling screens
    - editing: use strokes to delete, move, group, ...
    - history: save, restore, annotate
    - widget inference / recognition
Research Question of the Day

- **SILK: Specifying behaviors**
  - behavior = association of a screen object with another screen
  - specifying a behavior
    - put relevant screens in the “storyboard”
    - draw arrow from object to screen

- **SILK: Component recognition**
  - Infer / recognize components based on stroke-based gestures
    - scrollbar = long thin box + small inner box
  - Difficulties
    - differ’nt strokes for differ’nt folks
    - different stroke(s) may produce same drawing, but may be recognized very differently!
    - requires that system be trained for specific user

Research Question of the Day

- **SILK: Component composition**
  - When a component is recognized...
    - does the new component contain or is it contained by another component?
    - is the new component near another component?
    - is the new component in a sequence of components of the same type?

GUI evaluation

- **Why evaluate interfaces?**
  - commercial? bad GUIs can drive away customers
  - time-critical? bad GUIs can distract users
  - safety-critical? bad GUIs can kill... no kidding!
- **Evaluate early & often**
Evaluation methods

- Empirical testing
- Usability walkthroughs
- Heuristic evaluation
- User modeling

Empirical testing

- The “classic” method
  - Give users the interface & set of tasks
- Collect various measures from users
  - (more) subjective data
    - questionnaires
    - interviews
    - videotape
  - (more) objective data
    - time on task(s)
    - learning rate
    - low-level data: mouse movements, clicks, speech, gestures, eye movements, ...

Empirical testing

- Advantages
  - user is very involved!
    - can directly comment on interface
    - suggestions can be integrated... easily??
  - can yield objective measures
    - e.g., time on task, time to learn
    - but what are the right measures?

Empirical testing

- Disadvantages
  - tester expertise?
    - developing questionnaires, interviews
    - analyzing subjective videotape
  - tester’s knowledge of system?
    - testers & developers often not the same people
  - expense of new study?
    - usability labs need lots of equipment --> money
  - time required?
    - with the difficulty of analysis & integration of study,
      testing takes more than a short time
Usability walkthrough

- Gather group of users, designers, & human factors experts
- Interface is presented to group
  - prototype, storyboard, pen & paper, ...
- Freeform exploration of interface
  - ideas about usability, questionnaire, etc.
- More structured exploration:
  Cognitive walkthrough
  - “walk through” various scenarios with interface, playing the part of user performing a task

Usability walkthrough

- Advantages
  - again, users are involved
    - along with experts --> powerful interaction!
  - don’t need a full prototype
- Disadvantages
  - time consuming (esp. cognitive walkthrough)
  - coordination of all personnel + users?

Heuristic evaluation

- Developed by Jakob Nielsen
- 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
  - inspect various dialogue elements
  - compare with list of usability principles
  - consider other principles that come to mind
- Usability principles
  - Nielsen’s “heuristics”
  - supplementary list of category-specific heuristics
    - competitive analysis & user testing of existing products
- Use violations to redesign / fix problems

Thanks to James Landay @UC Berkeley for the foundation of these slides!
Heuristics (original)

- H1-1: Simple & natural dialog
- H1-2: Speak the users’ language
- H1-3: Minimize users’ memory load
- H1-4: Consistency
- H1-5: Feedback
- H1-6: Clearly marked exits
- H1-7: Shortcuts
- H1-8: Precise & constructive error messages
- H1-9: Prevent errors
- H1-10: Help and documentation

but we’ll look at the revised set...

Heuristics (revised)

- H2-1: Visibility of system status
  - keep users informed about what is going on
  - example: pay attention to response time
    - 0.1 sec: no special indicators needed
    - 1.0 sec: user tends to lose track of data
    - 10 sec: max. duration if user to stay focused on action
    - for longer delays, use percent-done progress bars

Heuristics (revised)

- H2-2: Match between system & real world
  - speak the users’ language
  - follow real world conventions
  - example: MacOS desktop
    - dragging disk to trash

Heuristics (revised)

- H2-3: User control & freedom
  - “exits” for mistaken choices, undo, redo
  - don’t force down fixed paths
  - example: “Wizards”
    - must respond to Q before going to next
    - for infrequent tasks
      - (e.g., modem config.)
    - not for common tasks
    - good for beginners
      - have 2 versions (WinZip)
Heuristics (revised)

- **H2-4: Consistency & standards**
  - especially important for impaired users... why?

- **H2-5: Error prevention**
  - make objects, actions, options, & directions visible or easily retrievable
  - example: pre-filling input fields
    - before dialing, asks for id & password
    - when connecting, asks again for id & password

- **H2-6: Recognition rather than recall**
  - make objects, actions, options, & directions visible or easily retrievable
  - example: pre-filling input fields
    - before dialing, asks for id & password
    - when connecting, asks again for id & password

- **H2-7: Flexibility and efficiency of use**
  - accelerators for experts (e.g., gestures, keyboard shortcuts)
  - allow users to tailor frequent actions (e.g., macros)

- **H2-8: Aesthetic and minimalist design**
  - no irrelevant information in dialogues
Heuristics (revised)

- H2-9: Help users recognize, diagnose, and recover from errors
  - error messages in plain language
  - precisely indicate the problem
  - constructively suggest a solution

Heuristics (revised)

- H2-10: Help and documentation
  - easy to search
  - focused on the user’s task
  - list concrete steps to carry out
  - not too large

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

Why multiple evaluators?

- Every evaluator doesn’t find every problem
- Good evaluators find both easy & hard ones (but different sets of easy & hard ones!!)
Heuristic evaluation

- Advantages
  - much faster than empirical testing
    - 1-2 hours each evaluator vs. days-weeks
  - doesn’t require interpreting user actions
- Disadvantages
  - user testing is far more accurate (by def.)
    - takes into account actual users and tasks
  - may miss problems & find “false positives”
- Good to alternate between HE & user testing
  - find different problems
  - don’t waste participants

Heuristic evaluation

- Studies of cost benefits [Nielsen’94]
  - benefit-cost ratio = 48 (!)
    - 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
- Studies of evaluators
  - 1 evaluator achieves poor results, only finds ~35% of usability problems
  - 5 evaluators find ~75% of usability problems
  - why not more evaluators??? 10? 20?
    - costs more, won’t find many more problems

User modeling

- The most complex, formal method
- Goal: Create a formal representation of user behavior to characterize, analyze it
- Tool: “Cognitive architectures”
  - psychological theory of thought / behavior
    - e.g., how quickly people forget information
  - computer representation of thought / behavior
    - can compute or simulate features of user behavior