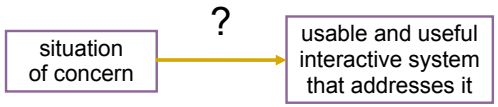
CPEN441: user interface design

HCI design process

Covered in this set…

- understanding the iterative stages of the HCI design process
- task-centred design
  - typical of S/W engineering process

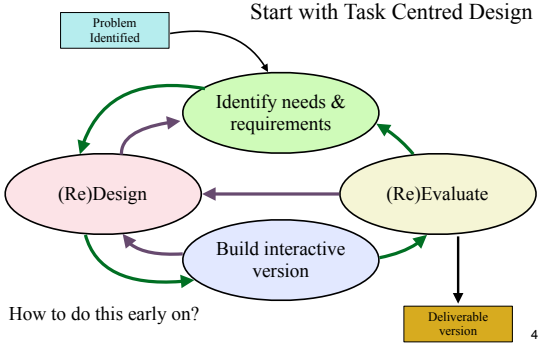
design process



- different design process models have been proposed
- we look at PRS, but there are others

PRS simple interaction design model

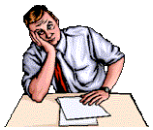
Start with Task Centred Design



How to do this early on?

task oriented system design

- HCI Requirements Analysis:
- exactly who uses the system to do what?



vs.

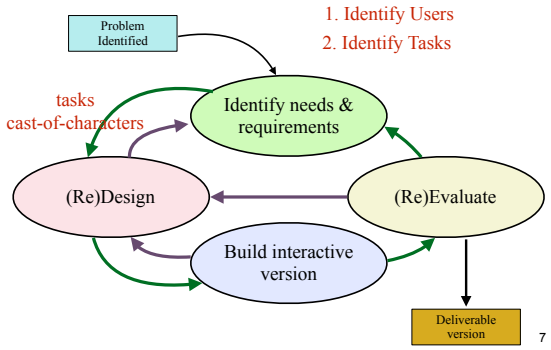
The User
a pretend person who will mould themselves to fit your system

Mary
a real person with real constraints trying to get her job done

the task oriented process

- phase 1: identification
  - identify specific users
  - articulate example realistic tasks
- phase 2: requirements
  - decide which of these tasks and users the design will support
- phase 3: design
  - base design representation and interaction sequences on these tasks
- phase 4: Evaluations with low-fi prototype
  - using your design, walk through the tasks to test proposed interface (you or another team)
  - use generic user model

PRS simple interaction design model



phase 1: identification

- contact with real people who will be potential users of system
  - identify specific end users: prototypical categories & extremes
- spend time with them discussing how the system might fit in
  - who would be willing to talk to you about this?
  - if you can't get them interested, who will actually buy/use your system?
- learn about the user's tasks
  - articulate concrete, detailed examples of tasks they perform or want to perform that your system should support
  - routine
  - infrequent but important
  - infrequent and incidental

phase 1: identification (cont)

- if there are no real users or tasks…
  - think again, there probably are!

Jeff Hawkins, the inventor of the Palm Pilot, was said to have carried a small block of wood around in his shirt pocket … As various everyday situations arose, he would take out the block of wood and imagine how he would use the device.[1]

[1]see Sato and Salvador, interactions 6(5)

…the same technique can be used to evoke a response from **expected** end-users

- if all else fails…
  - describe your expected set of users, and expected set of tasks
  - these will become your 'assumed users and tasks' that can be verified or modified later

good task examples:

1. say **what** the user wants to do but **does not say how** they would do it
   - no specific assumptions made about the interface
   - can be used to compare different design alternatives in a fair way

2. are very **specific**
   - says exactly what the user wants to do
   - specifies actual items the user would eventually want to input (somehow)

already showed example of hierarchical task descriptions:
- one way to describe a task

good task examples:

3. describe a complete job
   - **not just a list** of simple independent goals!
   - forces designer to consider **how interface features will work together**
   - contrasts how **information is carried through the dialog**
     - where does information come from? where does it go?

4. say who the users are *Cast of Characters technique*
   - design success strongly influenced by what users know
   - name names, if possible
   - reflect real interests of real users
   - helps find tasks that illustrate functionality in a person's real work context

"Cast of characters/Personas"

- imagine a set of individual users who span the possibilities of those you expect
  - ➡try to base them on people you have real experience with
- give them:
  - ➡ names, personalities, jobs, hobbies, lifestyles, needs and interests
- imagine each member of your cast using your interface, and what he/she would want from it.

good task examples:

5. are evaluated
   - circulate descriptions to users, and rewrite if needed
     → ask users for
     omissions corrections clarifications suggestions

6. as a set, identify a broad coverage of users and task types
   - the typical 'expected' user: typical routine tasks
   - the occasional but important user: infrequent but important tasks
   - the unusual user: unexpected or odd tasks

Identify **dependencies** among tasks

"**task objects**" are **resources** required by tasks
- artifacts (files, lists, databases)
- people (special expertise, authority, or knowledge)
- other processes, equipment or tasks

most tasks **focus** on a single resource
- task cannot be accomplished without the resource
- once resource is available the task can be completed

processes have **multiple dependencies**
- the focus shifts as different tasks are performed
- activity is suspended when resources are not available

Identify signs of task dependencies:

joint use of task objects by different tasks
e.g. access to shared files or databases

communication between people

may be direct (phone call) or indirect (memo)

synchronization

with real-world physical and mechanical processes

suspension

blocking when resources (information, people, real-world processes) are not available

Organizational approach: hierarchical **task** description

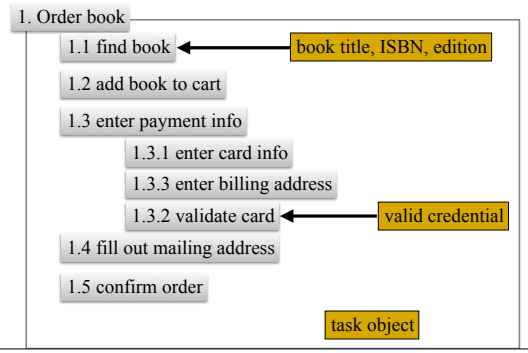
a sequence of steps makes up a task

- steps are linearly ordered
- steps can be carried out one after-the-other
- all steps contribute to the **parent goal** of the task

tasks may be **subtasks**

- subtasks may have a **subgoal**
- subtasks can be carried out one after-the-other
- notation can be either textual or graphical
- group steps under task goal

Example HTD



phase 2: requirements

- which user types will be accommodated?
  - designs can rarely handle everyone!
- discussion includes why particular users are included / excluded
- which (sub-) tasks will be addressed by the interface?
  - designs can rarely handle all tasks
  - requirements listed in terms of how they address tasks
    - absolutely must include: …
    - should include: …
    - could include: …
    - exclude: …
- discussion includes why items are in those categories
  - requirements are design-independent! (specs are design-specific)

example for **task** walkthrough: Cheap Shop Catalog Store

old way:

- people shop by browsing paper catalogs scattered around the store.
- when a customer sees an item she wants, she enters its item code from the catalog onto a paper form.
- customers give this form to a clerk, who brings the item(s) from the back room to the front counter.
- customers then pay for the items they want.

Item code	Amount
323066 667	1

developing task examples: Cheap Shop

- TASK: *at Cheap Shop, people browse a catalog and then order goods.*

- task example 1: *note: design-independent (not a scenario)*

  Fred, who is caring for his demanding toddler son, buys a stroller (red is preferred, but blue is acceptable), pays for it in cash, and uses it immediately.

  Fred is a first-time customer to this store and has little computer experience

developing task examples: Cheap Shop

- TASK: *at Cheap Shop, people browse a catalog and then order goods.*

- task example 2:

  an elderly arthritic woman is price-comparing the costs of a child's bedroom set, consisting of a wooden desk, a chair, a single bed, a mattress, a bedspread, and a pillow.

  she takes the description and total cost away with her, to check against other stores.

  two hours later, she returns and decides to buy everything but the chair.

developing task examples: Cheap Shop

•TASK: *at Cheap Shop, people browse a catalog and then order goods.*

•task example 3:

a "Cheap Shop" clerk, the sole salesperson in the store, is given a list of 10 items by a customer who does not want to use the computer.

the items are:

4 pine chairs, 1 pine table, 6 blue place mats, 6 "lor" forks, 6 "lor" table spoons, 6 "lor" teaspoons, 6 "lor" knives, 1 "tot" tricycle, 1 red ball, 1 "silva" croquet set

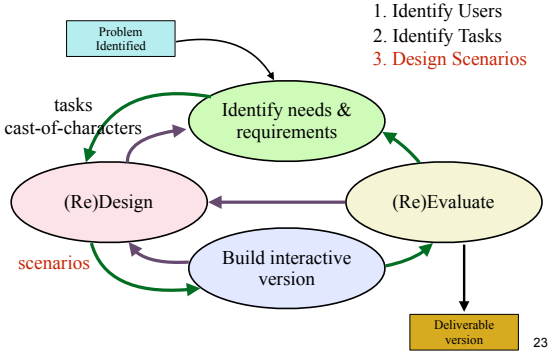
after seeing the total, the customer decides to take all but the silverware, and then adds 1 blue ball to the list.

the customer then changes his mind about paying by credit card, and decides to pay cash. The customer wants the items delivered to his home the day after tomorrow.

while this is occurring, 6 other customers are waiting for the salesperson.

PRS simple interaction design model

1. Identify Users
2. Identify Tasks
3. Design Scenarios



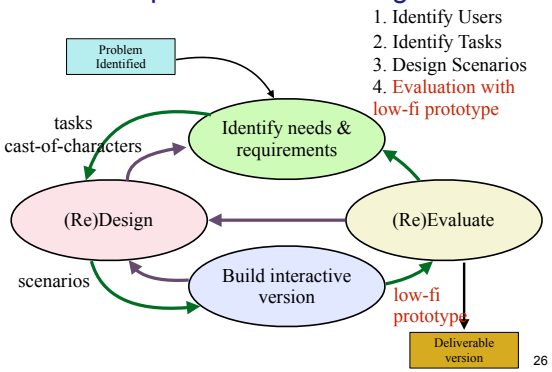
phase 3: design as scenarios

- develop designs to fit specific users and tasks
- use **task examples** to:
  - make design candidates concrete & debug them
  - consider how design features work *together* to help a person accomplish real work
  - consider the real world contexts of real users
- use **design scenarios** to consider how each task is handled by this design:
  - what the user would see / do step-by-step when performing the task
- **a scenario is design-specific**:
  it shows how a (design-independent) **task** would be performed given a particular design

“scenarios”

- informal narrative of that expands what, why and how of activities and tasks that people are doing
  - ➡usually tied to personas
- provide a specific example of a person progressing through the problem space
- provides a common story that whole team uses to communicate how people solve specific problems
- has expanded to be a common strategy for feature requirements in s/w eng (i.e. stories)

PRS simple interaction design model

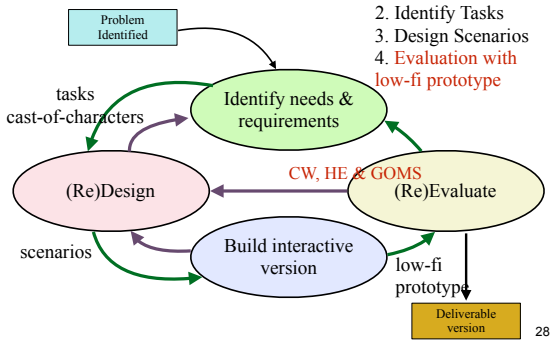


4. Evaluation

low-fi prototyping from scenarios

PRS simple interaction design model

1. Identify Users
2. Identify Tasks
3. Design Scenarios
4. Evaluation with low-fi prototype



the task centred process

- phase 1: identification
  - identify specific users
  - articulate example realistic tasks
- phase 2: requirements
  - decide which of these tasks and users the design will support
- phase 3: design
  - base design representation and interaction sequences on these tasks
- phase 4: Evaluations with low-fi prototype
  - using your design, walk through the tasks to test proposed interface (you or another team)
  - use generic user model

Task Analysis/Evaluation with User Model

eval with user *model* rather than actual users:

i. Cognitive Walkthrough
- using personas

iii. Heuristic Analysis
- using another team

ii. GOMS: Goal, Operators, Methods and Selection rules
- determine how long operation will take
- need to know the sequence of operations
  * Keystroke level analysis

eval i: cognitive walkthrough (CW)

- task focused
- good for developing / debugging an interface
- process:
  - select one of the task examples
    - for each user's step/action in the task and using your current interface prototype:
      - ask the 4 CW questions
      - ask: is there a believable story that motivates the user's actions?
      - if you locate a problem, mark it & pretend it has been repaired
- See example on website and more on youtube.

developing task examples: Cheap Shop

- TASK: *at Cheap Shop, people browse a catalog and then order goods.*

- task example 1: *note: design-independent (not a scenario)*

  Fred, who is caring for his demanding toddler son, buys a stroller (red is preferred, but blue is acceptable), pays for it in cash, and uses it immediately.

  Fred is a first-time customer to this store and has little computer experience

Cognitive walkthrough for prototype

Q1: Will the correct action be evident to the user?

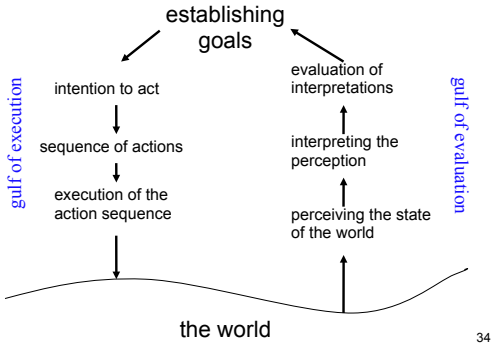
Q2: Will user connect the correct action with their goal?

Q3: Will user interpret the system's response to the chosen action correctly, i.e., will user know from this response, whether they made the right or wrong choice?

Q4: *Will user's mental models be affected? Will new concepts be added, or existing concepts lost?*

Remember: you are trying to get into mind of character using a persona. Don't be the designer.

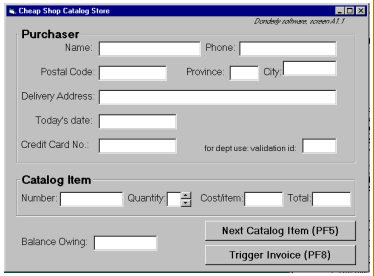
What Mental Models Tell the User



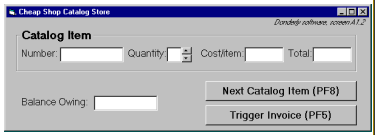
Cheap Shop

- 1st prototype of a new design

Screen 1



Screen 2



Cheap Shop: 1st prototype specifications

to create an order

- on screen 1, shoppers enter their personal information and their first order
- text is entered via keyboard
- the tab or mouse is used to go between fields.

further orders

- shoppers go to the 2nd screen by pressing the Next Catalog Item button

order completion

- shoppers select 'Trigger Invoice'.
- the system automatically tells shipping and billing about the order
- the system returns to a blank screen #1

to cancel order

- shoppers do not enter input for 30 seconds (as if they walk away)
- the system will then clear all screens and return to the main screen

input checking

- all input fields checked when either button is pressed.

task walkthrough template

1st: break task down into steps

Task number: ____

User _____

Description of Step	correct action evident?	connect action to goal?	interpret the response?	*mental models affected?*	Comment / solution

task walkthrough template

1st: break task down into steps

Task number: ____

User _____

Description of Step	Does the user have the knowledge/training to do this?	Is it believable that they would do it? Are they motivated?	Comment / solution

limitations

- tasks almost always embody a process
(even though they are not specific to a particular interface)
  - may be hard to produce 'pure' 'system' or 'process' independent tasks
  - may not encourage consideration of alternate ways to do tasks
  - may be impossible to find someone who actually does the task

an alternative to tasks: goal-oriented system design

• articulates user goals rather than how they want to do them
  - goal: - a desired end condition
    - tend to be stable
  - task: - an intermediate process needed to achieve the goal
    - may change as technology / work patterns change
• designer analyzes goals, looking for solutions and how to satisfy them
→ may result in different task / task sequence which could be better

eval ii: heuristic evaluation

- for: identifying (listing & describing) problems with existing prototypes (any kind of interface)
- **not for:** coming up with radically new solutions
- additional advantages:
  - contributes valuable insights from objective observers
  - only needs low-fi prototype or better
- disadvantages:
  - reinforces existing design - better solutions might exist
  - not very repeatable

eval ii: heuristic evaluation

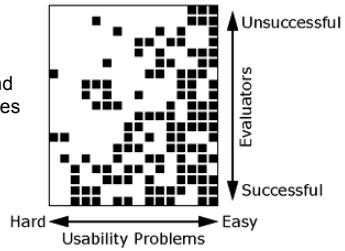
• what's required: a good model of the proposed interface (e.g., at least a paper prototype) + a list of design heuristics to be applied

• who does it: team of 3 to 5 experienced, objective people who **aren't on the design team**.

• general idea: independently check compliance with usability principles ("heuristics")
  - each evaluator works with interface *alone*
  - different evaluators will find different problems
  - evaluators aggregate findings afterwards

why multiple evaluators?

- every evaluator doesn't find every problem
- good evaluators find both easy & hard ones



heuristic evaluation process

- evaluators go through UI several times
  - inspect various elements (screen, dialogs, etc.)
  - compare with list of usability principles
  - consider other principles/results 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

Nielsen's heuristics (original - 1994)

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

commonly used list has evolved

see video discussion on website

PRS list of heuristics

H2-1: visibility of system status
H2-2: match between system & the real world
H2-3: user control & freedom
H2-4: consistency and standards
H2-5: help users recognize, diagnose & recover f/ errors
H2-6: error prevention
H2-7: recognition not recall
H2-8: flexibility and efficiency of use
H2-9: aesthetic and minimalist design
H2-10: help and documentation

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
- **each evaluator** produces list of problems
  - explain problem w/reference to heuristic or other info
  - be specific and **list each problem separately**
- tips:
  - be respectful but critical
  - let your client decide whether to ignore a problem
  - look especially for what's *not* there

severity & extent scales

severity scale:
3 = Major usability problem
2 = Minor usability problem
1 = Cosmetic problem or nuisance issue

extent scale:
3 = Widespread
2 = Several places
1 = Single case

aggregating and returning the evaluation

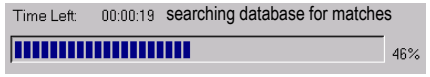
- evaluation team meets and compares results
- through discussion and consensus, each violation is documented and categorized in terms of severity
- violations are ordered in terms of severity

•→ **combined report goes back to design team.**

H2-1: visibility of system status

• keep users informed about what is going on

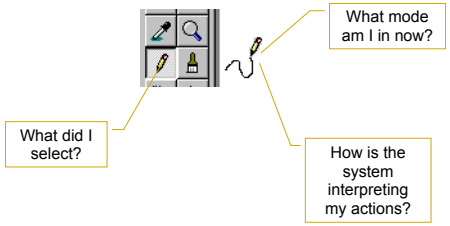
- example: consider system response time (user must wait)
  - 0.1 sec: no special indicators needed, why?
  - 1.0 sec: user starts to lose track of data, objects, etc
  - 10 sec: max duration if user to stay focused on action
  - for longer delays, use percent-done progress bars



H2-1: visibility of system status

•keep users informed about what is going on

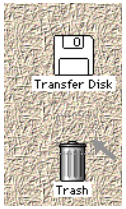
- appropriate visible feedback



H2-2: match between system & real world

- **speak the users' language**
- **follow real world conventions**

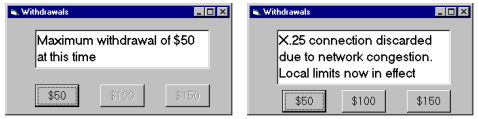
- (bad) example: Mac desktop
  - dragging disk to trash
  - this should delete it, not ***eject*** it!



H2-2: match between system & real world

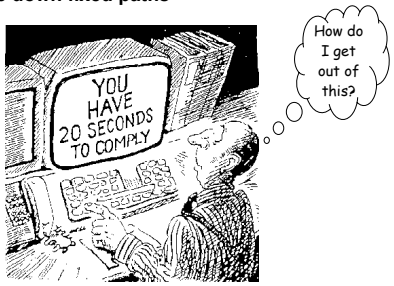
- **speak the users' language**
- **follow real world conventions**

- e.g. withdrawing money from a bank machine



H2-3: user control & freedom

"exits" for mistaken choices, undo, redo
don't force down fixed paths



H2-3: user control & freedom

"exits" for mistaken choices, undo, redo
don't force down fixed paths

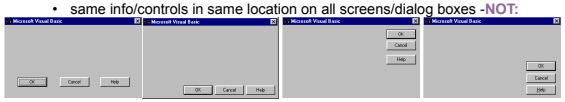
strategies:

- 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)

H2-4: consistency & standards

•consistency of effects → **predictability**
- same words, commands, actions should always have the same effect in equivalent situations

•consistency of language and graphics
- same info/controls in same location on all screens/dialog boxes -**NOT:**



- same visual appearance across the system (e.g. widgets)
  - e.g. NOT different scroll bars in a single window system!

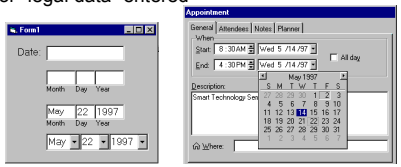
•consistency of input
- require consistent syntax across complete system

H2-5: help users recognize, diagnose, and recover from errors

- **error messages in plain language**
- **precisely indicate the problem**
- **constructively suggest a solution**

H2-6: error prevention

- **try to make errors impossible**
  - modern widgets: only "legal commands" selected, or "legal data" entered



  - provide reasonableness checks on input data

e.g. upon entering order for office supplies:

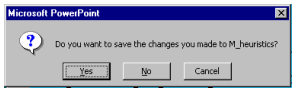
"5000 pencils is an unusually large order. Do you really want to order that many?"

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 behavior
    - usually due to inattention
  - often arises from similarities of actions

types of slips

- **capture error**
  - frequent response overrides [unusual] intended one
  - occurs when both actions have same initial sequence
    - change clothes for dinner → find oneself in bed (William James, 1890)
    - confirm saving of a file when you don't want to delete old version



types of slips

- **description error**
  - intended action has too much in common with others possible

  e.g. 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
    - move file to trash instead of to folder
- **loss of activation**
  - forgetting the goal while carrying out the action sequence

  e.g. start going to a room and forget why by the time you get there
    - navigating menus/dialogs, can't remember what you are looking for
- **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

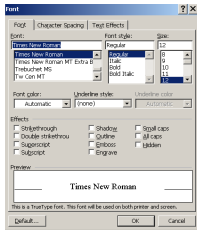
designing for slips

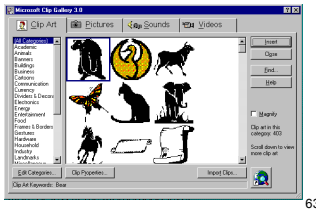
- **general rules**
  - prevent slips before they occur
  - detect and correct slips when they do occur
  - user correction through feedback and undo
- **examples**
  - mode errors
    - have as few modes as possible (preferably none)
    - make current mode highly apparent
  - capture errors
    - instead of confirmation, allow users to undo actions

    e.g. Mac trash can can be opened and "deleted" file taken back out
  - loss of activation
    - if system knows goal, make it explicit
    - if not, allow person to see path taken
  - description errors
    - in icon-based interfaces, make sure icons are not too similar,
    - check for reasonable input, etc.

H2-7: recognition rather than recall

- **computers good at remembering things, people aren't!**
  - menus, icons, choice dialog boxes vs. cmd lines, field formats
  - relies on visibility of objects to the user (but less is more!)



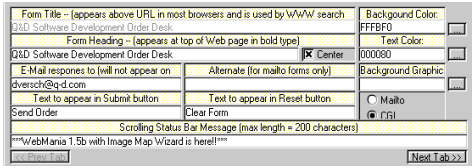


H2-8: flexibility and efficiency of use

- **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
    - go to desired location directly, avoiding intermediate nodes
  - history systems
    - WWW: ~60% of pages are revisits

H2-9: aesthetic and minimalist design

- **no irrelevant information in dialogues**
- bad example:



H2-10: help and documentation

- **help is not a replacement for bad design!**
- simple systems: walk up and use; minimal instructions
- most other systems:
  - feature-rich
  - some users want to become “expert” rather than “casual” users
  - intermediate users need reminding, plus a learning path
- many users do not read manuals
- usually used when users are panicked & need help NOW
  - need online documentation, good search/lookup tools
  - online help can be specific to current context
- sometimes used for quick reference
  - syntax of actions, possibilities...
  - list of shortcuts …

remember: how to perform Heur Eval

1. design team supplies scenarios, prototype; need 3-5 evaluators
2. each evaluator **independently** produces list of justified, rated problems by stepping through interface and applying heuristics at each point … use heuristics list & severity rating convention
3. team meets and compiles report that organizes and categorizes problems

summary: heuristic evaluation

- advantages
  - the "minimalist" approach
    - general guidelines can correct for 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 and end users
- problems:
  - principles must be applied intuitively and carefully
    - can't be treated as a simple checklist
    - subtleties involved in their use
  - doesn't necessarily predict users/customers' overall satisfaction
  - may not have same "credibility" as user test data
    - a solution: include design team & developers in usability evaluation

summary: heuristic eval, cont.

- research result:
  - 4-5 evaluators usually able to identify 75% of usability problems
  - user testing and usability inspection have a large degree of non-overlap in the usability problems they find (i.e., it pays to do both)
- cost-benefit:
  - usability engineering activities often expensive / slow; but some can be quick / cheap, and still produce useful results
  - usability inspection turns less on what is "correct" than on what can be done within development constraints
  - ultimate trade-off may be between doing *no* usability assessment and doing *some* kind

eval iii: GOMS

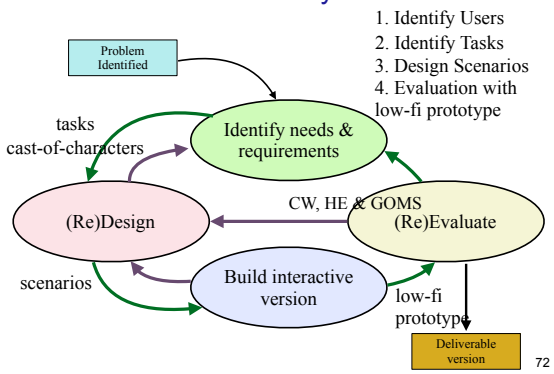
- Goals are the end-state
- Operators are basic actions available
- Methods are sequences of operators
- Selection rules are invoked when there is a choice

- estimate time for each action or decision to get task completion times
- good when you have clear task space and interface prototype

Keystroke-Level Analysis

- estimate speed of execution of task
  - When no selection rules needed
- provide times for each keystroke operation, i.e.:
  - K = key press [0.08-1.2]
  - P = point with mouse to target
    - Fitts law: P ∝ log Distance/Area; typically [0.8-1.5]
  - H = homing hands on keyboard
  - D(nd,ld) = draw straight line segments; 0.9nd+.16ld
  - M = mental prep time; 1.35
  - R(t) = response time by system if user has to wait
- Add all times up for a keyboard task to estimate completion time.

Summary:



summary

- understand **iterative stages** of design
- **task-centered** system design: how to
  - users and stakeholders -> personas
  - develop task examples -> scenarios
  - use them to motivate designs
  - create low-fi prototypes
  - evaluate designs through:
    - cognitive walkthrough (CW)
    - heuristic evaluation (HE)
    - GOMS