EVALUATION OF PROTOTYPES
USABILITY INSPECTION "DISCOUNT" METHODS

690A- Advanced Methods in HCI

Prof. Narges Mahyar

Slides from Prof. Joanna McGrenere and Dr. Leila Aflatoony
Includes slides from Prof. Karon MacLean and Jessica Dawson

TODAY

- MAP assessment [20 min]
- Cognitive walkthrough [20 min]
- Heuristic evaluation [15 min]
- CommunityClick demo [5 min]
- In class activity [15 min]

t e f d
THE INSTITUTE FOR
TEACHING EXCELLENCE & FACULTY DEVELOPMENT

MAP

Midterm Assessment Process

VOLUNTARY, CONFIDENTIAL, AND FORMATIVE

1. **Form a *small group* with your classmates (no more than 4-5 per group). Elect *one recorder*.**

2. **THEN, go to the following url to complete the assessment with your group**

tinyurl.com/compsci690a

LEARNING GOALS

- explain why cognitive walkthrough and Heuristic evaluation are considered discount usability methods
- outline the general procedure for conducting a heuristic evaluations and a cognitive walkthrough know how to apply heuristics
- describe the pros/cons of cognitive walkthroughs and Heuristic evaluation, and explain when it is an appropriate choice of evaluation method
  - give examples of what each is an **appropriate choice**

DISCOUNT USABILITY ENGINEERING

cheap (thus 'discount')

- no special labs or equipment needed
- doesn't need to involve users *directly*
- the more careful you are, the better it gets

fast

- on order of 1 day to apply
- standard usability testing may take a week

easy to use

- can be taught in 2-4 hours

TYPES OF DISCOUNT METHODS

cognitive walkthrough: "mental model"

- assesses "exploratory learning stage"
- what mental model does the system image facilitate?
- done by non-experts and/or domain experts

heuristic evaluation: "fine tune"

- fine-tunes the interface (hi-fi prototypes; deployed systems)
- HCI professionals apply a list of heuristics while simulating task execution
- targets broader use range (including expert)

COGNITIVE WALKTHROUGH

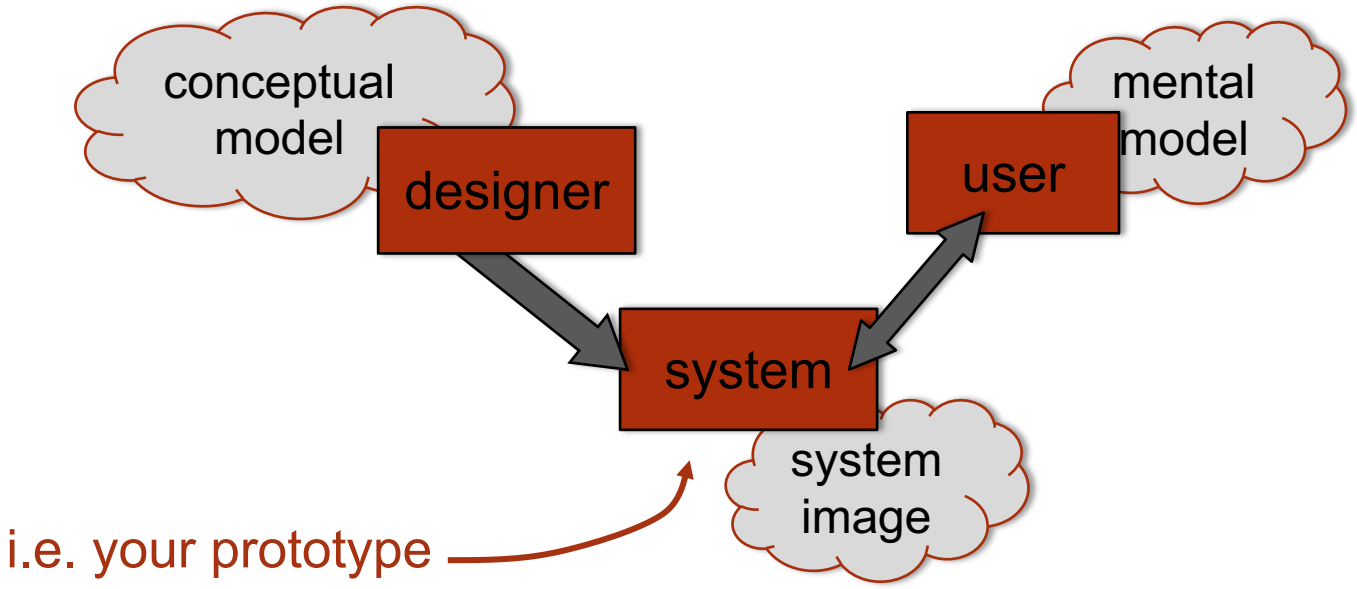
WHAT IS A COGNITIVE WALKTHROUGH?

- cognitive walkthroughs are used to evaluate a product's usability.
- **TEST** conceptual model/interface support for mental models though **task examples**: task + design = **scenario**
- use a "**walkthrough**" to evaluate a **scenario**

CW SIMULATES MENTAL MODEL DEVELOPMENT

Assessing…

- is the conceptual model an effective one?
- does the interface design communicate the conceptual model?
- how well does it support forming a good mental model?



COGNITIVE WALKTHROUGH EXPLORATORY LEARNING

what for: developing / debugging an interface, *without accessing users (which is expensive)*

tests: how well

1) interface design
2) underlying conceptual model aligns with/sets up the user's mental model

not for: assessing performance at highly skilled, frequently performed tasks; or finding radically new approaches

HOW TO CONDUCT A WALKTHROUGH EVALUATION?

start: with a scenario
task examples + design ➔ *scenario*

process:

1) break **task** down into **user actions** (expected system response)

2) perform each step ON the existing **interface** and ask:

Q1: will the user know what to do?

Q2: will the user see how to do the action?

Q3: will the user correctly understand the system response?

3) if you locate a problem, mark it & pretend it has been repaired; then go on to next step.

COGNITIVE WALKTHROUGH

possible outputs:

- loci & sources of confusion, errors, dead ends
- estimates of success rates, error recovery;
  performance speed less evident
- helps to figure out what **activity sequences** could or should be

what's required:

- task examples: **design-independent** descriptions of tasks that representative users will want to perform.
- a prototype to **provide a design**.

who does it: [theoretically] anyone – usually design team members or expert outside analysts.

- can use real users . . . but this makes it a lot less 'discount'

COGNITIVE WALKTHROUGH: BASIC STEPS

Step I. Generate "correct", **intended steps** to complete a task.

Select a task to be performed and write down all the 'user actions', and expected "system responses".

(a) can they find correct sequence(s) in current version?
***use high-level directives**:* correct user action = *''enter amount of food for pet feeder to dispense"*

(b) are there mental-model problems even if they use exactly the right sequence?
***get very specific**: correct user action = "type '36g' into the text entry box in the middle of the screen*

COGNITIVE WALKTHROUGH: BASIC STEPS

Step II. Carry out steps, *simulating the mindset of your intended user*, and note your success OR failure on a log sheet.

for each step:

Q1: ask yourself if user knows what to do?
- are they trying to produce this effect? do they have enough info? etc.

Q2: explore – will the user see how to do the step?
- look for the needed action? is it visible? it is obvious how to perform the step?

Q3: interpret – will the user correctly understand the system response?
- Is the feedback understandable? Will the interpretation be correct?

Note: even with an error, user may have progressed if error became apparent. Distinguish this from when user is left with a misunderstanding.

COGNITIVE WALKTHROUGH:
TWO APPROACHES TO INSTRUCTING PERSON(S) DOING CW

Approach (a): participant follows the **pre-prepared steps** and assess according to expected actions/system response

- at each step, assess using the questions usually best you can do with a paper/low-fidelity prototype (unless it is very complete, has many paths)
- approach you will probably want to use in project

Approach (b): give the CW participant **ONLY the higher level directive**(s).

- E.g., “create an event note with the following attributes. . . ”
- more exploratory; still use Q1-3 to assess for each step they take
- BUT - the steps he/she takes might diverge from the list you made – note them down on another action-list sheet. These points should trigger further analysis
- usually most effective higher fidelity prototypes or released systems

COGNITIVE WALKTHROUGH:
WHAT KINDS OF PROBLEMS SHOULD I RECORD?

in a CW you may note may kinds of problems, e.g.,

- e.g., problems with particular steps
- problems moving between steps
- larger problems that involve lots of steps
- larger problems that hint at deeper problems with conceptual model/design
- small problems that might only apply to unusual users
- other kinds of problems that just become apparent while using interface, etc.

make note of these as as appropriate

- if you do a lot of CWs, you may develop your **own template** for noting problems that works for you

COGNITIVE WALKTHROUGH:
HOW DO I BECOME GOOD AT DOING CWS?

1. when you're new to CWs, it's easy to assume to the user will know what to do if YOU know what to do
   - force yourself to imagine what the user might not know
2. when asking the questions at each step:
   - really think about what the user could be thinking. . .
   - consider the impact of misconceptions or mistakes that they could have made earlier!
3. perform lots of them!
   - you'll get better at figuring out what to focus on with practice

COGNITIVE WALKTHROUGH:
WHAT DO I DO AFTER THE CW?

CWs can be done in teams or individually

- aggregate and discuss problems
    - possibly found over more than one CW
- prioritize problems based on severity, likelihood

THEN:

- iterate and fix as required
    - decide on which you can/will address
    - iterate on conceptual model and/or interface design
- OR write up a report/recommendations → design team
    - if you're not the one(s) doing the designing

HEURISTIC EVALUATION

HEURISTIC EVALUATION

what for:

- identifying (listing & describing) problems with existing prototypes (any kind of interface); for any kind of user, new or proficient

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
- 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

SCOTT KLEMMER

https://www.coursera.org/lecture/human-computer-interaction/heuristics-understanding-flwJl

HOW TO PERFORM A HEURISTIC EVALUATION

1. design team supplies scenarios, prototype, list of heuristics; need 3-5 evaluators: train in method if non-expert
   - single evaluator catches ~35% of the usability problems
   - five evaluators catch ~75%

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

INDIVIDUALS VS. TEAMS

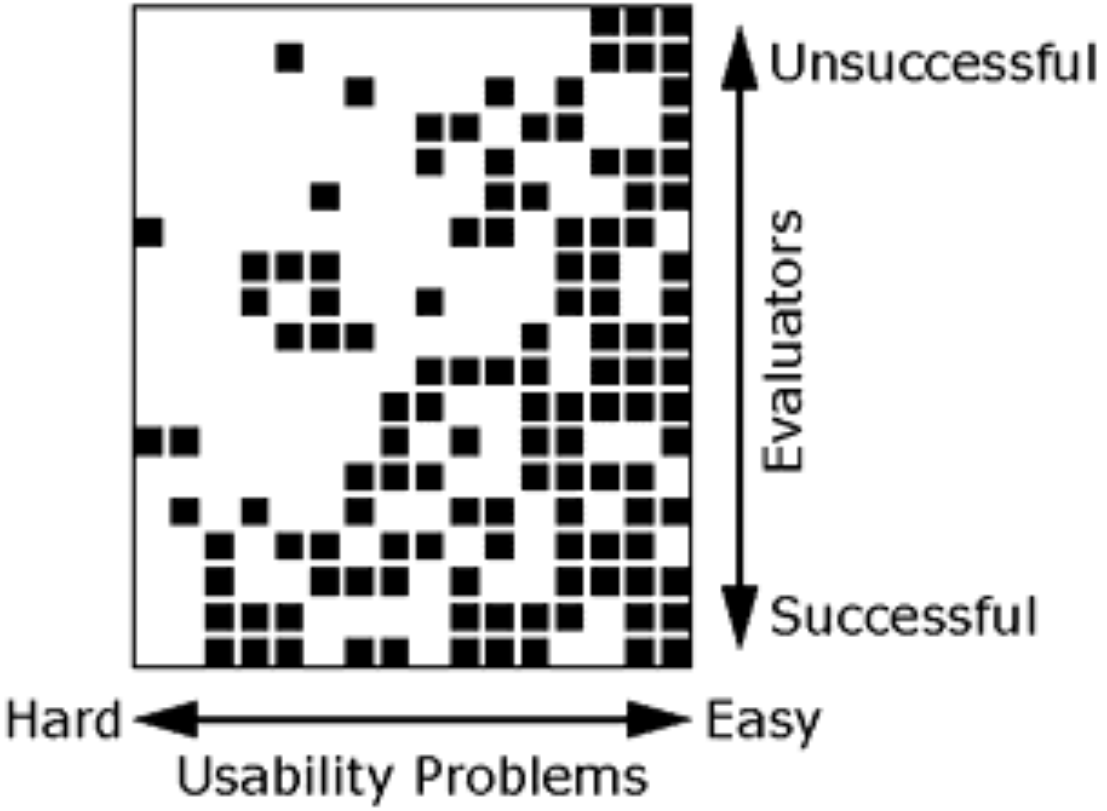
Nielsen recommends individual evaluators inspect the interface alone.

Why?

- evaluation is not influenced by others
- independent and unbiased
- greater variability in the kinds of errors found
- no overhead required to organize group meetings

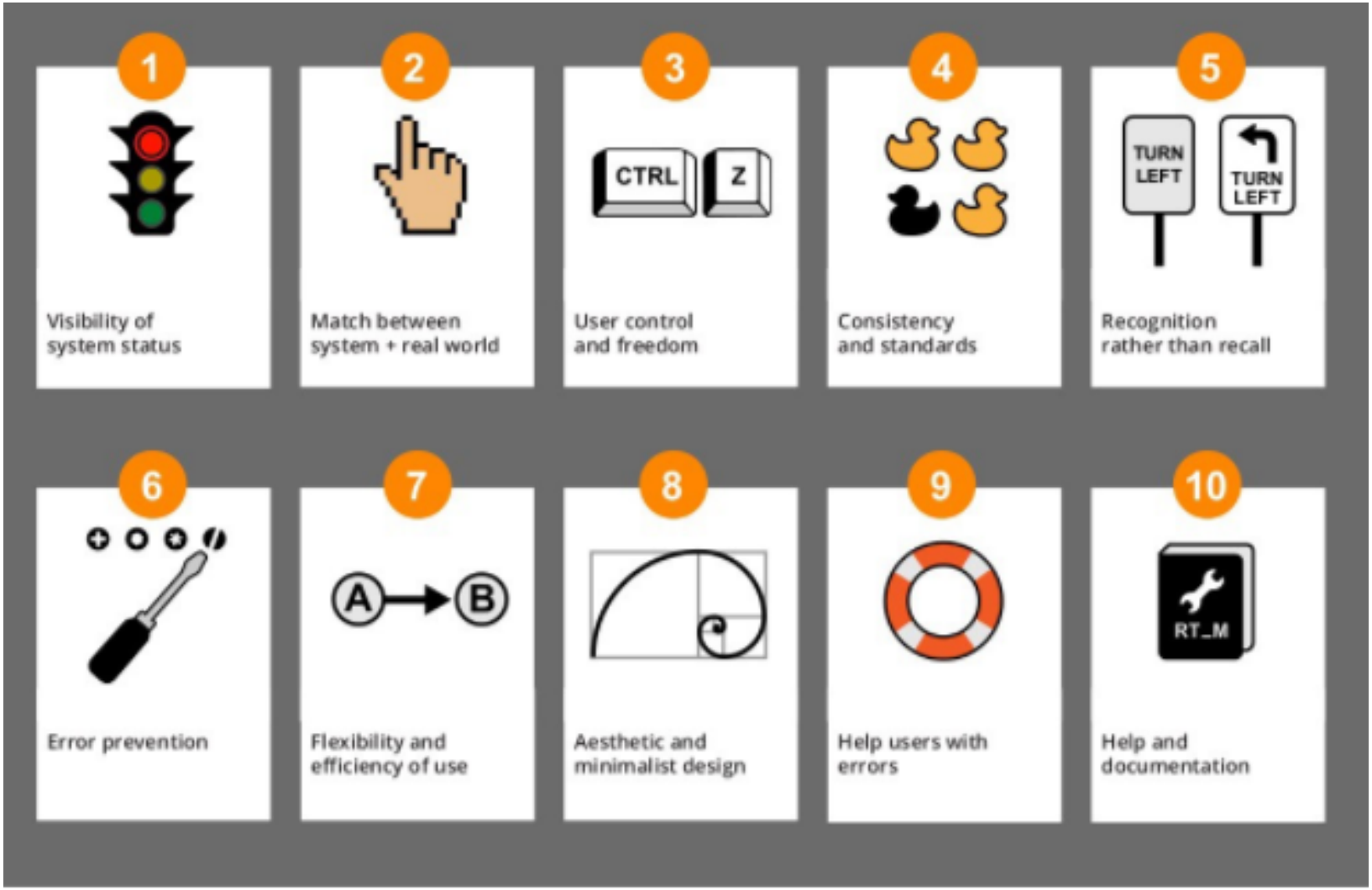
WHY MULTIPLE EVALUATORS?

- every evaluator doesn't find every problem
- proficient evaluators find both easy & hard (subtle) ones



ONE POPULAR LIST OF HEURISTICS (NIELSON, '93)

H1:	visibility of system status
H2:	match between system & the real world
H3:	user control & freedom
H4:	consistency and standards
H5:	recognition rather than recall
H6:	error prevention
H7:	flexibility and efficiency of use
H8:	aesthetic and minimalist design
H9:	help users recognize, diagnose & recover f/ errors
H10:	help and documentation


1
Visibility of system status
2
Match between system + real world
3
CTRL
Z
User control and freedom
4
Consistency and standards
5
TURN LEFT
TURN LEFT
Recognition rather than recall
6
Error prevention
7
A
B
Flexibility and efficiency of use
8
Aesthetic and minimalist design
9
Help users with errors
10
RT_M
Help and documentation

STEP 1: BRIEFING SESSION

get your experts together

- brief them on what to do, goals of system, etc.
- discuss heuristics to be applied

may also want to provide experts with:

- some examples of tasks
- descriptions of user personas
- simple instructions/guidance
    - especially if NOT a fully functioning system

STEP 2: INDIVIDUAL 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
- assign rating of severity to each violation

EVALUATION FORM

Example Heuristic Evaluation Form

Evaluator: _____ Prototype: _____ Date/Time: _____ Pg: ___ / ___

Heuristic violated	Description / Comment	Severity

SEVERITY RATINGS

each violation is assigned a severity rating

- many other methods of doing this

usually some combination of:

- frequency
- impact
- persistence (one time or repeating)

used to:

- help prioritize problems
- allocate resources to fix problems
- estimate need for more usability efforts

can be done independently by all evaluators or later as group prioritizes

EXAMPLE SEVERITY & EXTENT SCALES

one severity scale:

0 - don't agree that this is a usability problem
1 - cosmetic problem
2 - minor usability problem
3 - major usability problem; important to fix
4 - usability catastrophe; imperative to fix

one extent scale:

1 = single case
2 = several places
3 = widespread

STEP 3: AGGREGATING RESULTS & MAKING RECOMMENDATIONS

- **evaluation team** meets and compares results
- through discussion and consensus, each violation is documented and categorized in terms of severity, extent
- violations are ordered in terms of severity
    - e.g., use an excel spreadsheet (which can be sorted)
- combined report goes back to design team.

HEURISTIC EVALUATION

Advantages

- contributes valuable insights from objective observers
- the “minimalist” approach
    - general guidelines can correct for majority of usability problems
    - easily remembered, easily applied with modest effort
    - systematic technique that is reproducible with care.
- *discount* usability engineering
    - cheap and fast way to inspect a system
    - can be done by usability experts and rapidly-trained end users

HEURISTIC EVALUATION

problems:

- principles must be applied intuitively and carefully
    - can't be treated as a simple checklist
- heuristics can narrow focus on some problems at cost of others
- can reinforce existing design (not for coming up with radical ideas)
- doesn't necessarily predict users/customers' overall satisfaction
- may not have same "credibility" as user test data

COMBINING HE AND CW

➔ HCI practitioners often use a combination of both that might vary based on what they're trying to learn

- e.g., while doing a walkthrough for a task, apply the heuristics at each step in addition to the CW questions.

ACTIVITY:
THE PROTOTYPE DESIGN

conceptual model (simplified):

➔ Visualize community reactions to each proposal to give a quick overview with the ability to sort and filter for more exploration and drill down to the actual comments.

Proposals:

- have title, topics, comments, participants
- have positive, negative, and neutral sentiment and reactions

ACTIVITY:
GENERATING STEPS FOR CW

CommunityPulse (https://communitypulse.cs.umass.edu):

a visual analytic system that utilizes text analysis to extract important topics, emotions and sentiments from community comments and enables civic leaders to explore the comments at multiple levels of granularity.

WORK OUT STEPS FOR TASK EXAMPLE
(WITH 'CORRECT' ACTIONS FOR GIVEN INTERFACE)

1. Decides to use CommunityPulse
2. Sorts based on excited comments
3. Sorts based on angry comments
4. Sorts based on negative comments
5. Selects the proposal with the most angry comments
6. Goes back to the overview page
7. Selects two top proposals with the largest number of comments

ACTIVITY PART 1: WORK OUT STEPS FOR CW

work in pairs

1) follow steps from the task scenario + storyboard

- *use storyboard to help you understand order of steps/mapping to screens*
- *you might not always have enough info to determine what the correct user action should be, that's OK*

➔ *can guess based on your knowledge of how similar systems work OR skip it*

2) we will generate a set (with correct actions) – whole class

ACTIVITY PART 2: PERFORM THE COGNITIVE WALKTHOUGH

work in pairs

for each of the steps:

- ask yourselves each of Q1-Q3;
- if answer is NO for any questions:
    - write down the problem (possible solutions if you have ideas)
    - THEN assume it's fixed; go on to next step

DISCUSSION ON REQUIREMENT READINGS [20 MIN]

Get into group of 3-4 answering the following questions:

- What surprised you? or
- What you disagreed with?
- Others?

ON DECK…

Next class (Thursday) …

- Reading and researcher journal
- First prototypes

EXTRA SLIDES

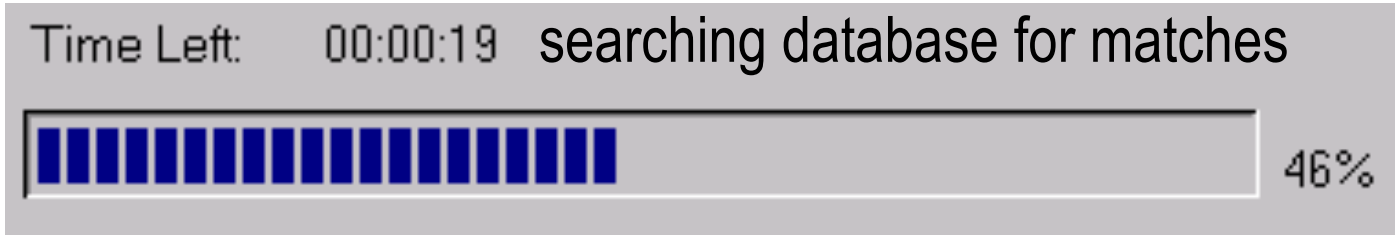
HEURISTICS

H1: VISIBILITY OF SYSTEM STATUS

The system should always keep users informed about what is going on, through (appropriate feedback within reasonable time)

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



H1: VISIBILITY OF SYSTEM STATUS

keep users informed about what is going on

- appropriate visible feedback

What mode am I in now?

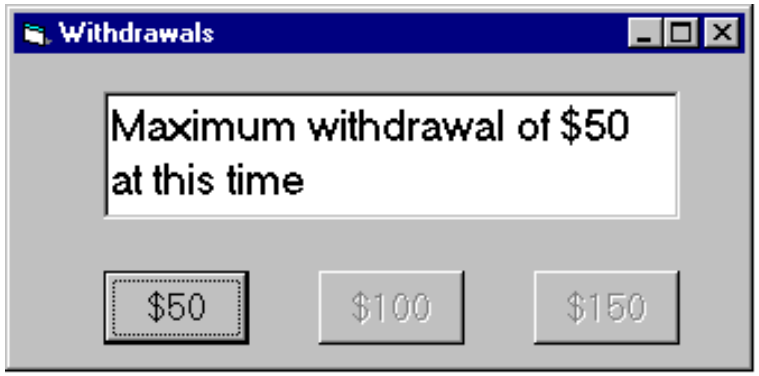
What did I select?

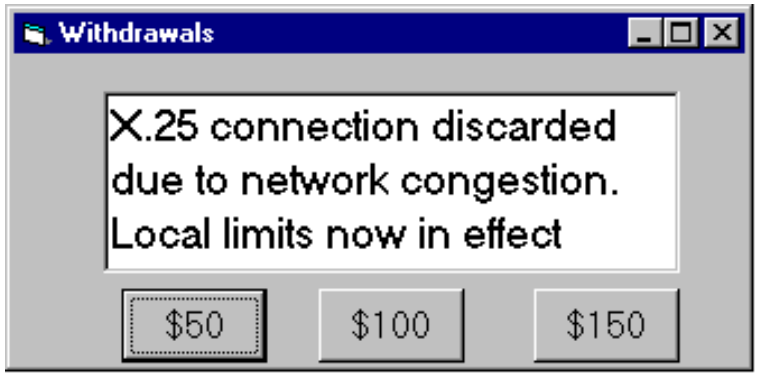
How is the system interpreting my actions?

H2: MATCH BETWEEN SYSTEM & REAL WORLD

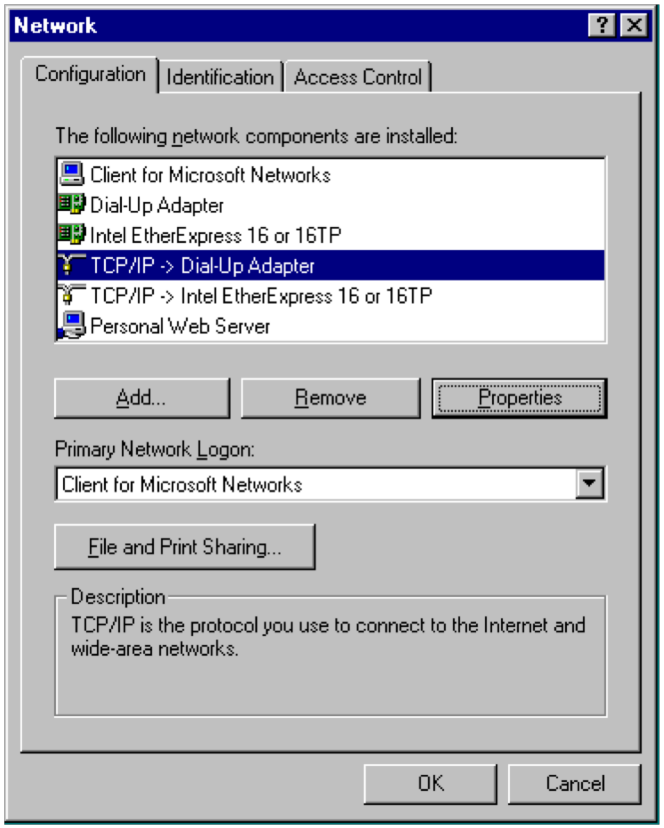
The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

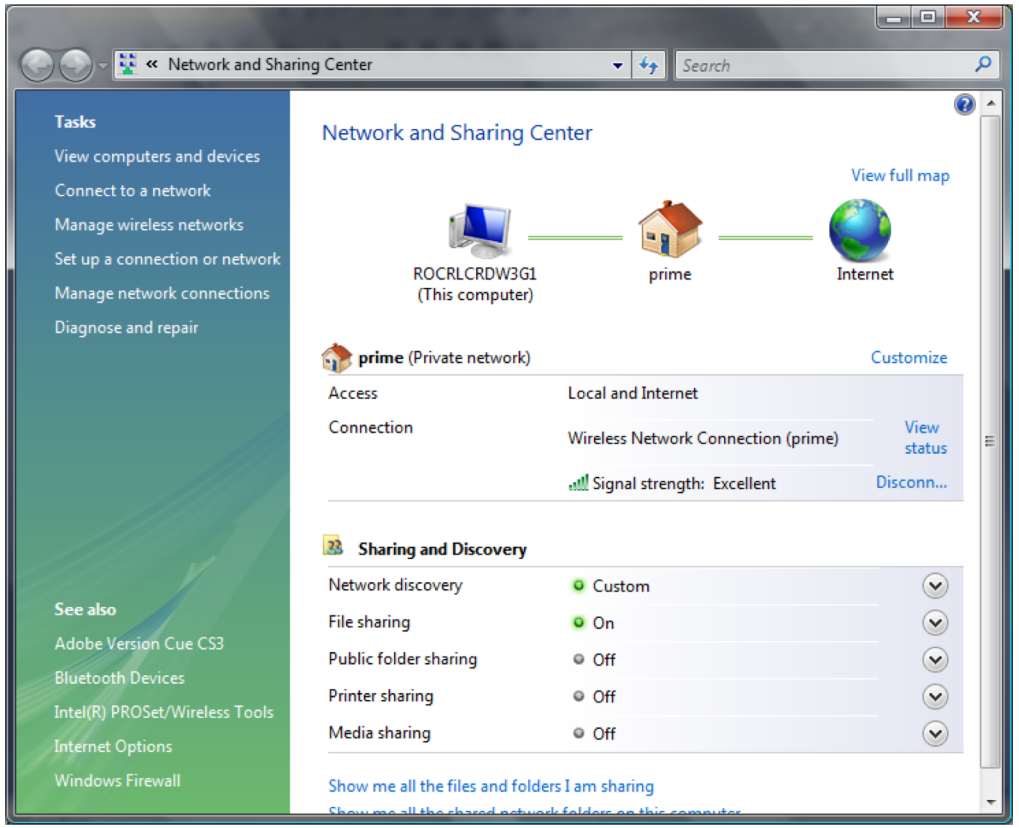
e.g. withdrawing money from a bank machine





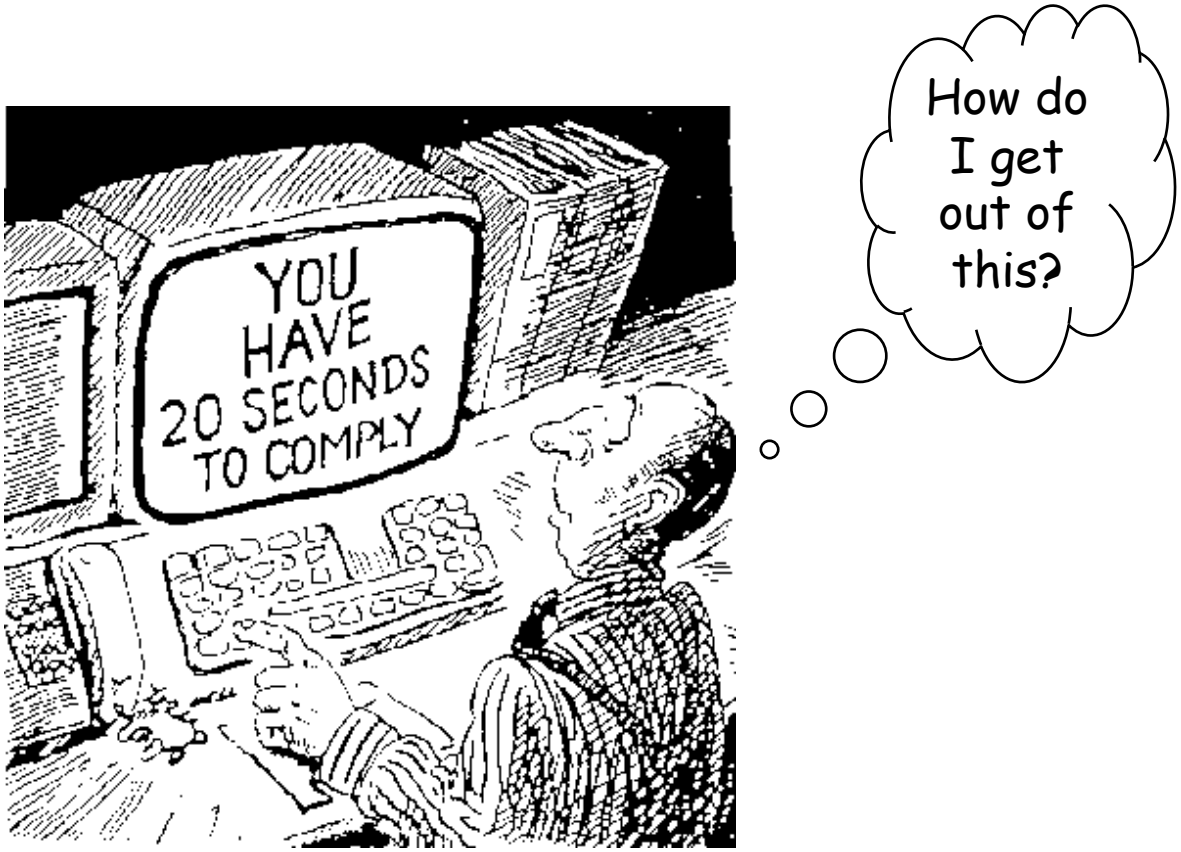
H2: MATCH BETWEEN SYSTEM & REAL WORLD





H3: USER CONTROL & FREEDOM

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.



H3: 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)

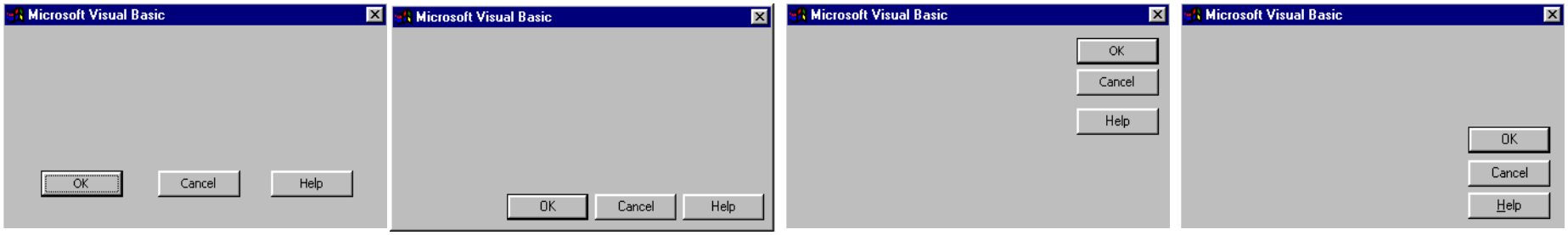
H4: CONSISTENCY & STANDARDS

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

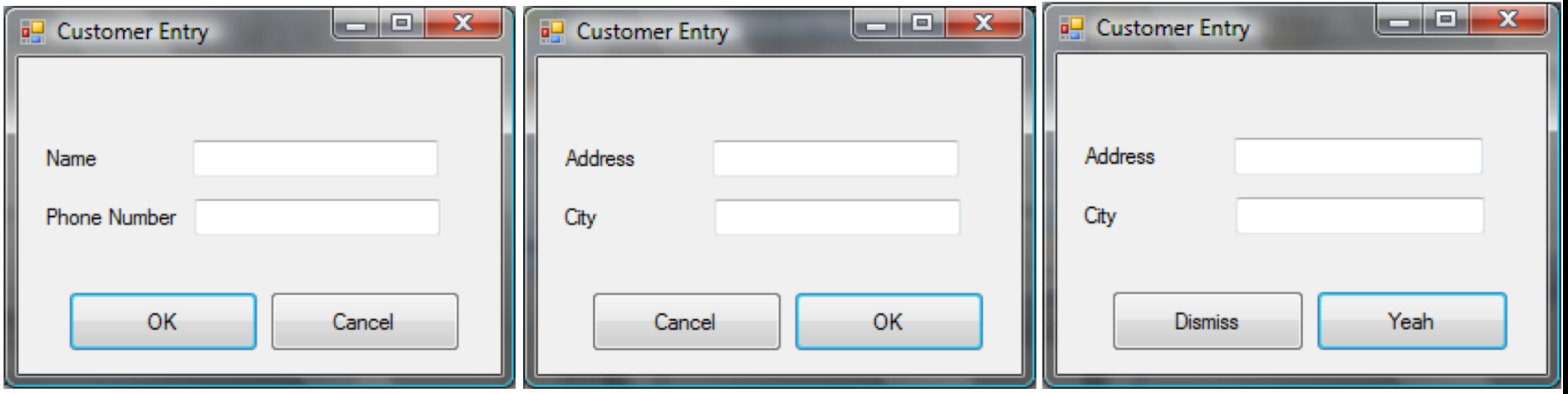
consistency of input

- require consistent syntax across complete system

H4: CONSISTENCY & STANDARDS

consistency of language and graphics

- same info/controls in same location on all screens/dialog boxes

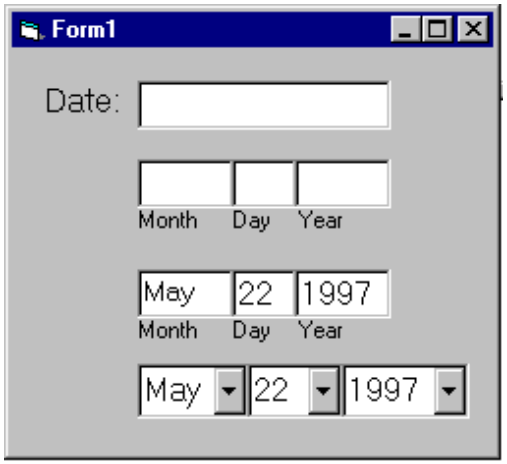


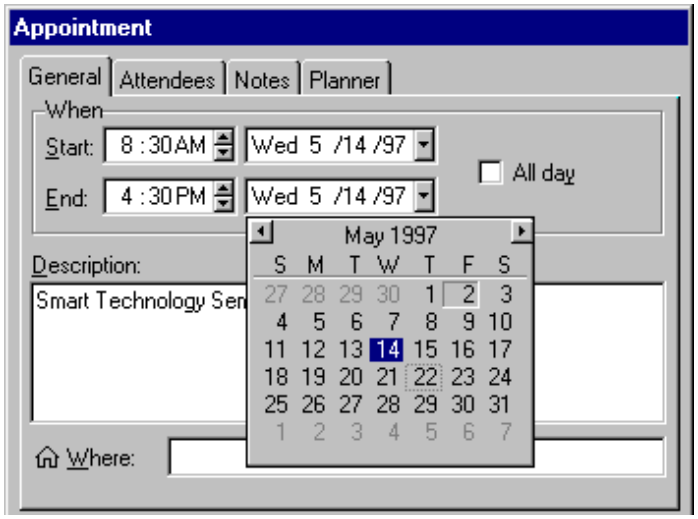
H5: ERROR PREVENTION

try to make errors impossible

- Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

modern widgets: only “legal commands” selected, or “legal data” entered





H5: ERRORS WE MAKE

mistakes

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

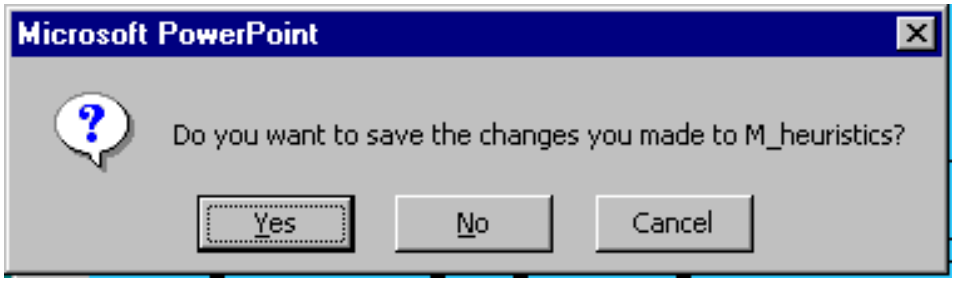
slips

- unconscious behavior 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

H5: TYPES OF SLIPS

capture error

- frequent response overrides [unusual] intended one
- occurs when both actions have same initial sequence
  - confirm saving of a file when you don't want to delete old version



H5: 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
  - but continue action to remember (or go back to beginning)!

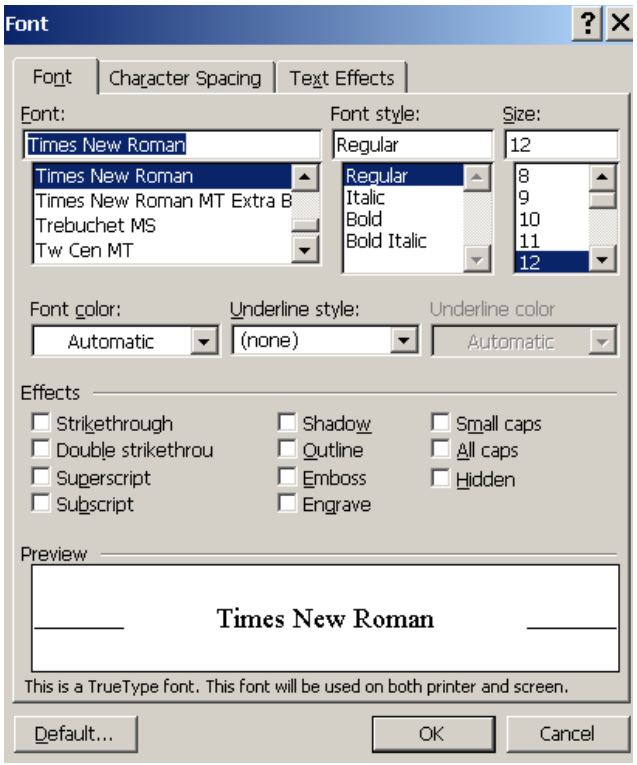
mode errors

- 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

H6: RECOGNITION RATHER THAN RECALL

computers good at remembering things, people aren't!

Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.



H7: FLEXIBILITY AND EFFICIENCY OF USE

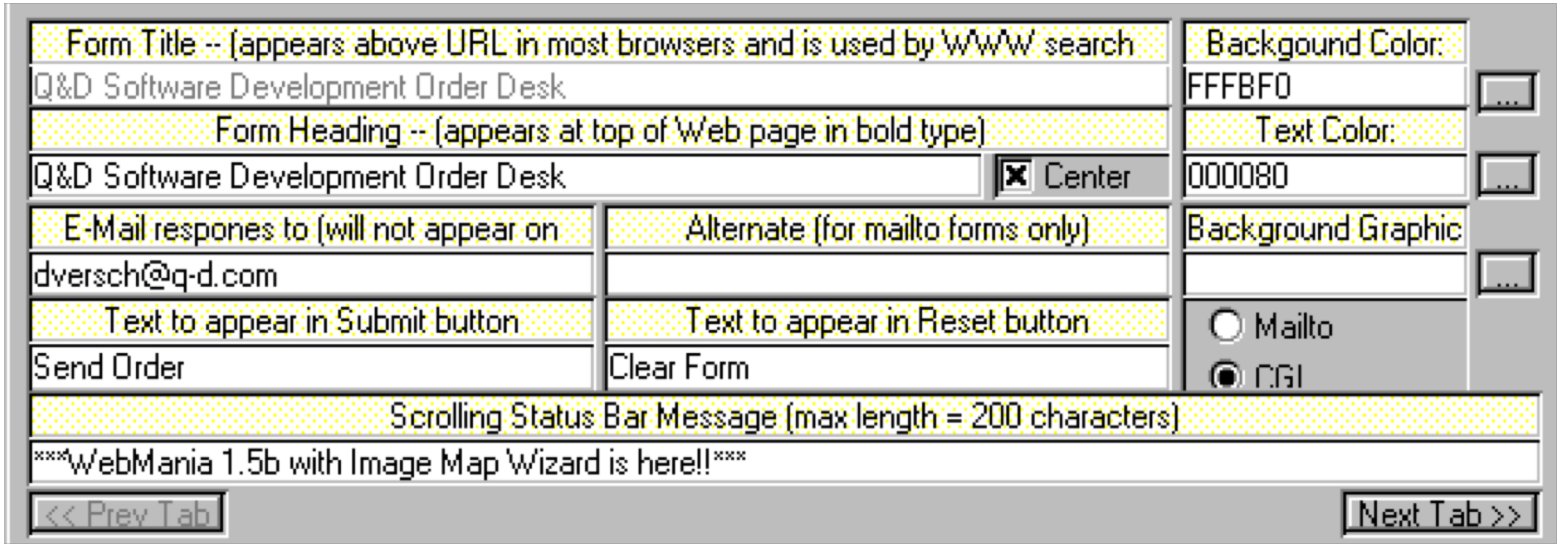
experienced users should be able to perform frequently used operations quickly

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

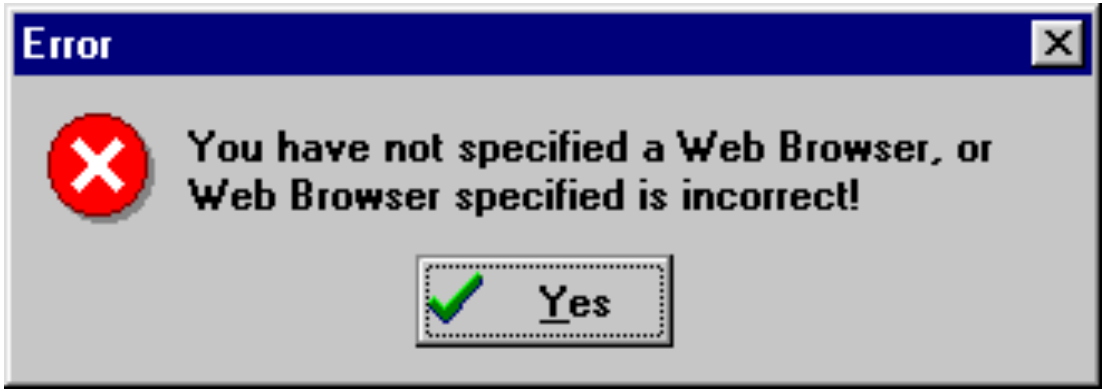
H8: AESTHETIC AND MINIMALIST DESIGN

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.



H9: HELP USERS RECOGNIZE, DIAGNOSE, AND RECOVER FROM ERRORS

Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.



H10: 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 …

H10: TYPES OF HELP

tutorial and/or getting started manuals

- short guides that people usually read when first encounter system
    - encourage exploration and getting to know the system
    - communicate conceptual material and essential syntax
- on-line “tours”, exercises, and demos
    - demonstrate very basic principles through working examples

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

H10: TYPES OF HELP (CONT'D)

reminders

short reference cards

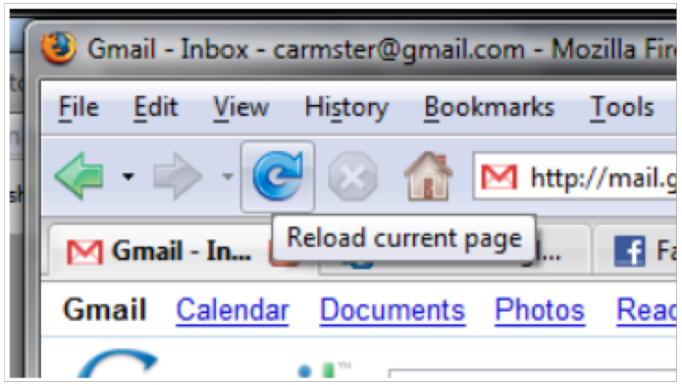
- expert user who just wants to check facts
- novice who wants to get overview of system's capabilities

keyboard templates

- shortcuts/syntactic meanings of keys; recognition vs. recall; capabilities

tooltips

- text over graphical items indicates their meaning or purpose



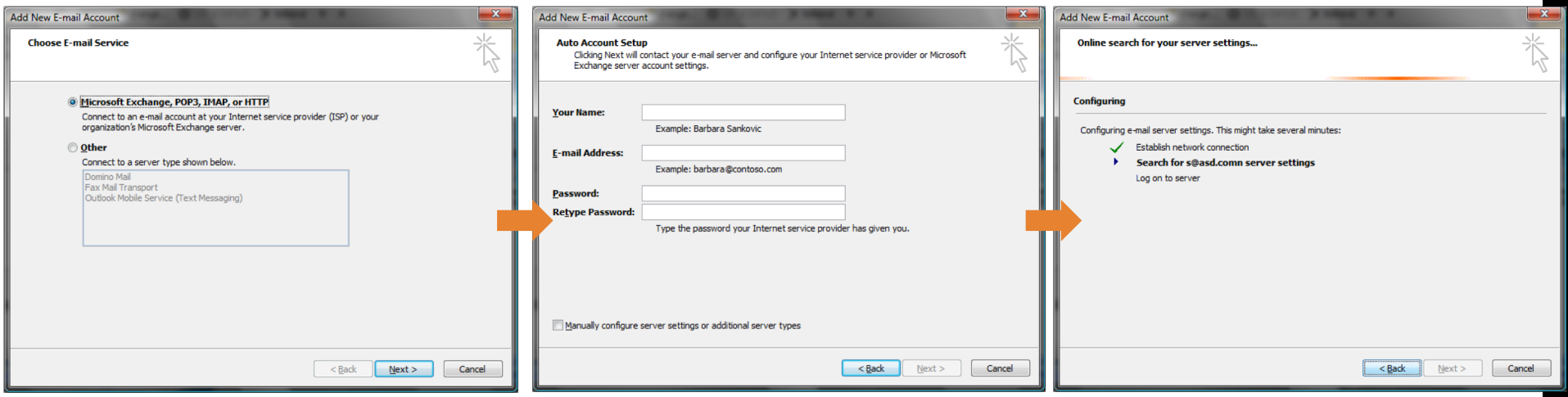
H10: TYPES OF HELP (CONT'D)

context-sensitive help

- system provides help on the interface component the user is currently working with
  - Macintosh "balloon help"
  - Microsoft "What's this" help

wizards

- walks user through typical tasks
- reduces user autonomy



REFERENCE

https://www.nngroup.com/articles/ten-usability-heuristics/