

# Introduction to HCI

## Evaluation of Prototypes Discount Methods

**Prof. Narges Mahyar**  
**UMass Amherst**

[nmahyar@cs.umass.edu](mailto:nmahyar@cs.umass.edu)

Courses, projects, papers, and more:

<http://groups.cs.umass.edu/nmahyar/>

# Today

- Reading discussion [5 min]
- Cognitive walkthrough [20 min]
- Heuristic evaluation [15 min]
- In class activity [15 min]
- Project discussion [15 min]

# 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

# 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)

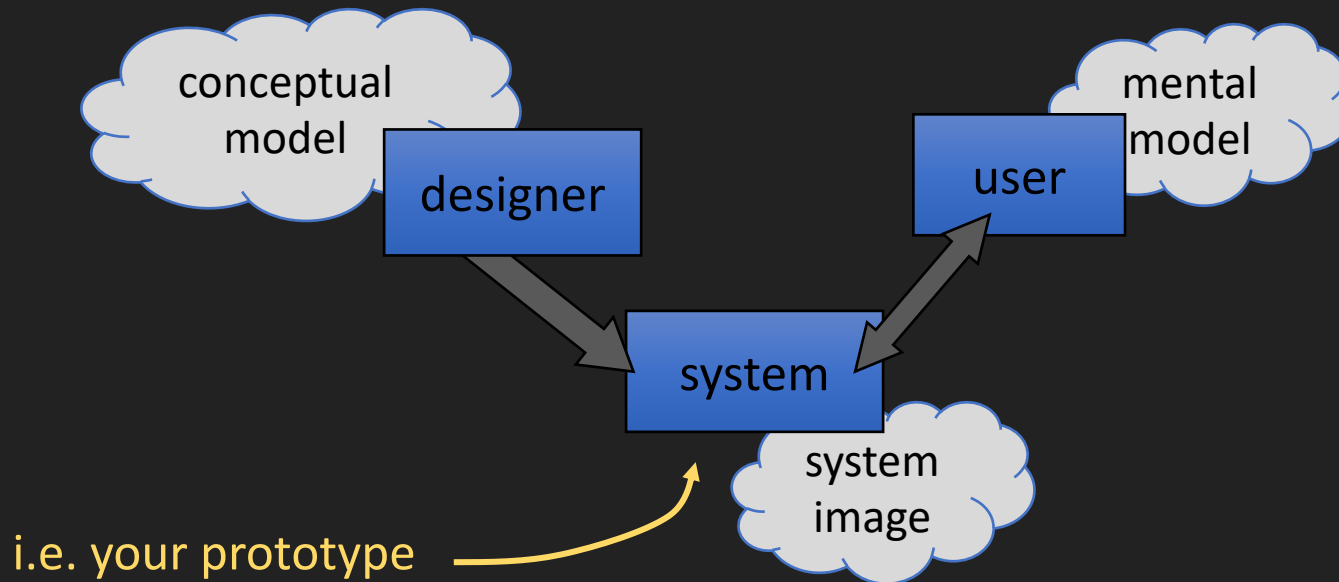
# Cognitive walkthrough

## What is a cognitive walkthrough?

- Cognitive walkthroughs are used to evaluate a product's usability.
- **Test** conceptual model/interface support for mental models through **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**
- 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 1.** 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 many kinds of problems, for example:
  - 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 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

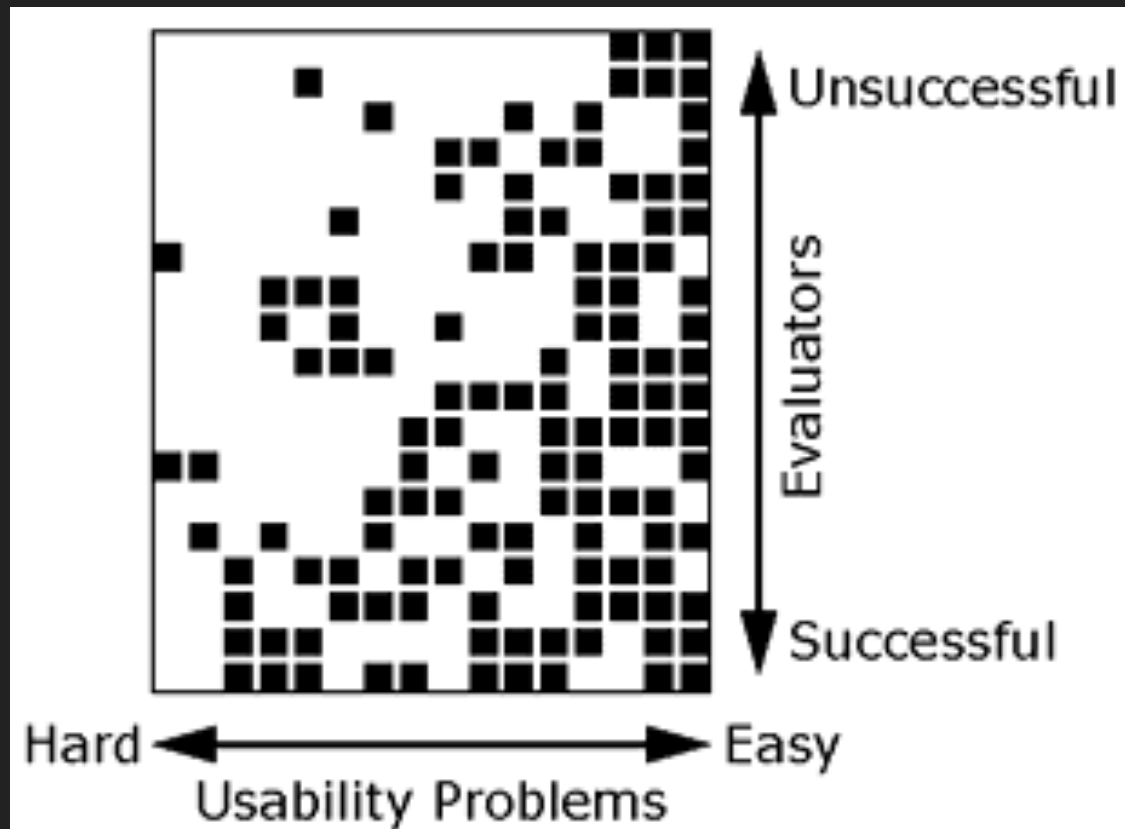
- Nielsen recommends individual evaluators inspect the interface alone.

## *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

H10: help and documentation



1



Visibility of system status

2



Match between system + real world

3



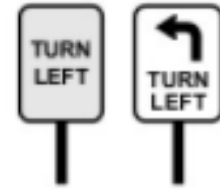
User control and freedom

4



Consistency and standards

5



Recognition rather than recall

6



Error prevention

7



Flexibility and efficiency of use

8



Aesthetic and minimalist design

9



Help users with errors

10



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: *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. Ron 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 with your group
- 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 walkthrough

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

# On deck...

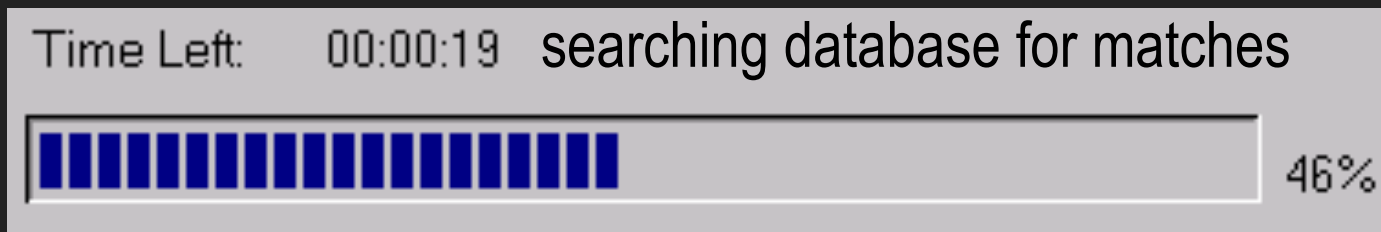
- Next class (Thursday) ...
- Readings

# Extra slides heuristics



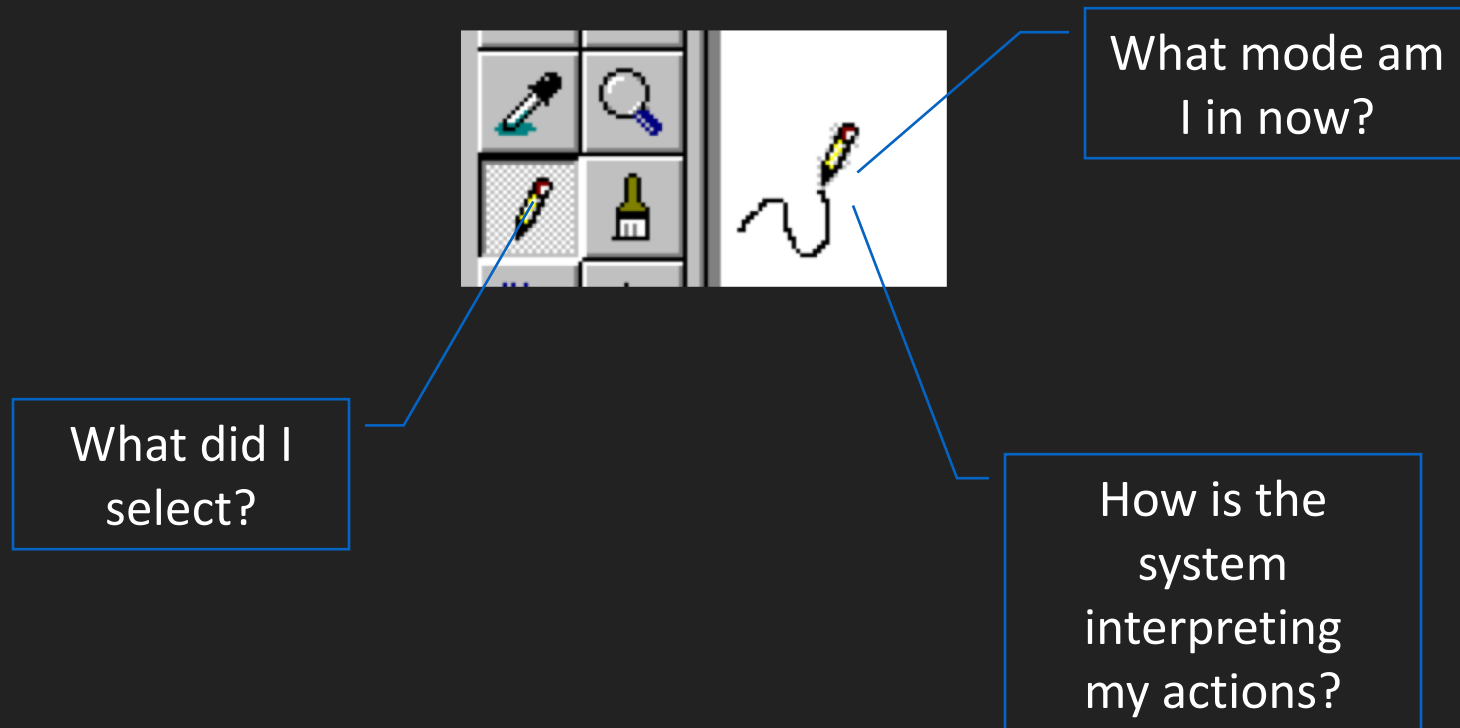
# H1: visibility of system status

- 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



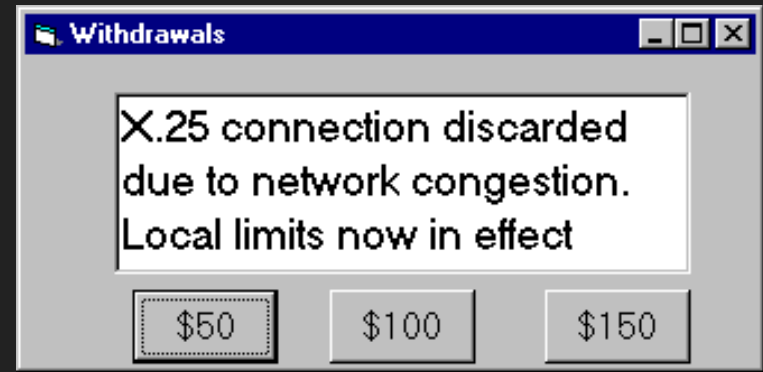
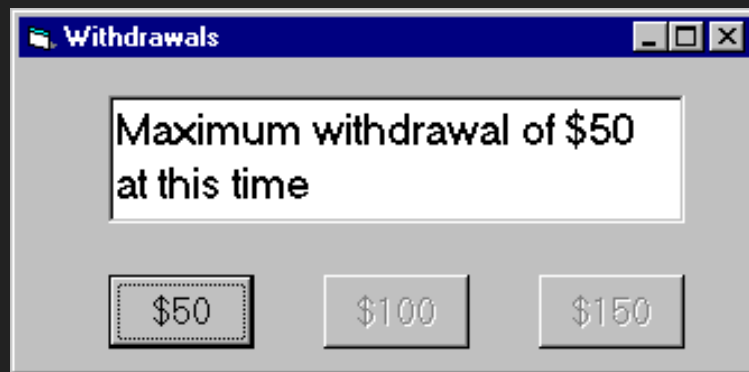
# H1: visibility of system status

- **keep users informed about what is going on**
  - appropriate visible feedback

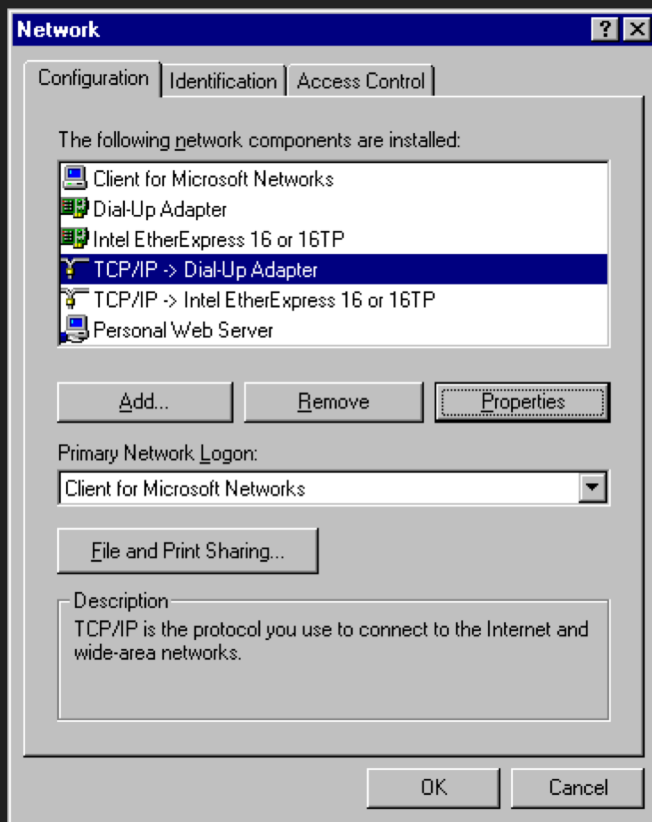


## H2: match between system & real world

- 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.



How do I get out of this?

## H3: user control & freedom

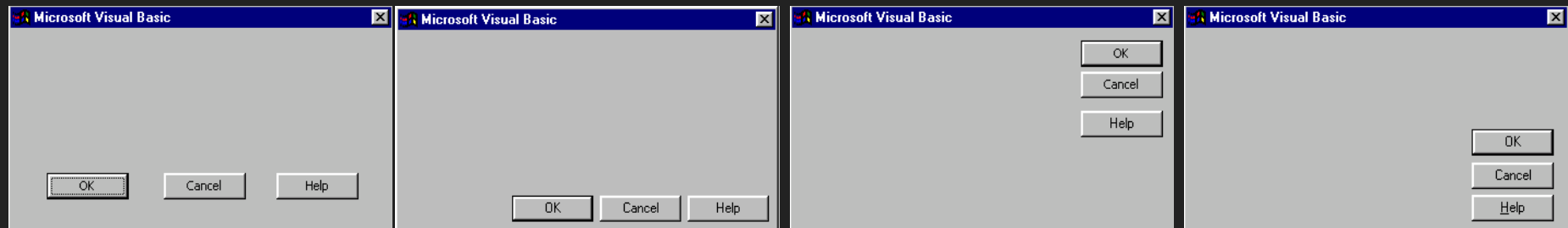
- “Exits” for mistaken choices, undo, redo
- Don’t force down fixed paths

### 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:**

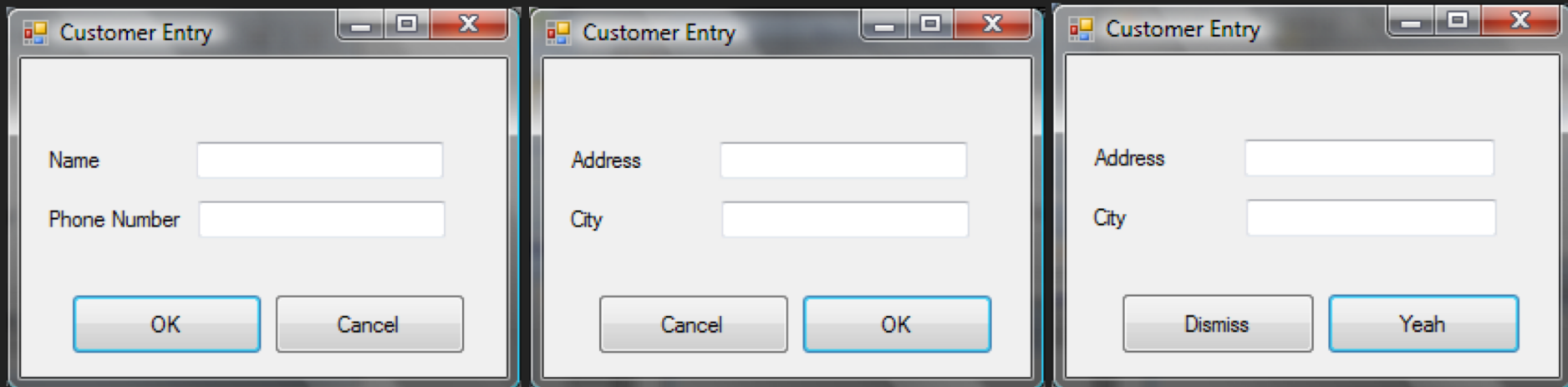


- same visual appearance across the system (e.g. widgets)
  - e.g. not different scroll bars in a single window system
- consistency of input
  - require consistent syntax across complete system

# H4: consistency & standards

consistency of language and graphics

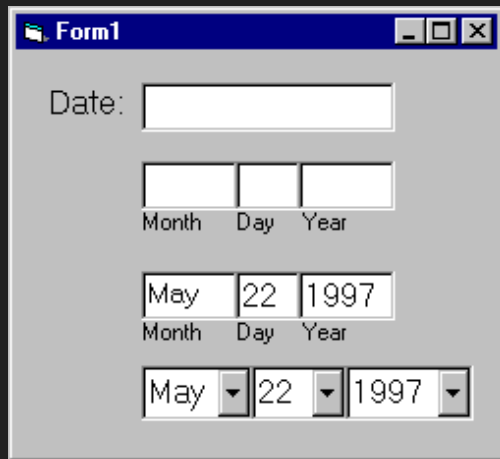
- same info/controls in same location on all screens/dialog boxes



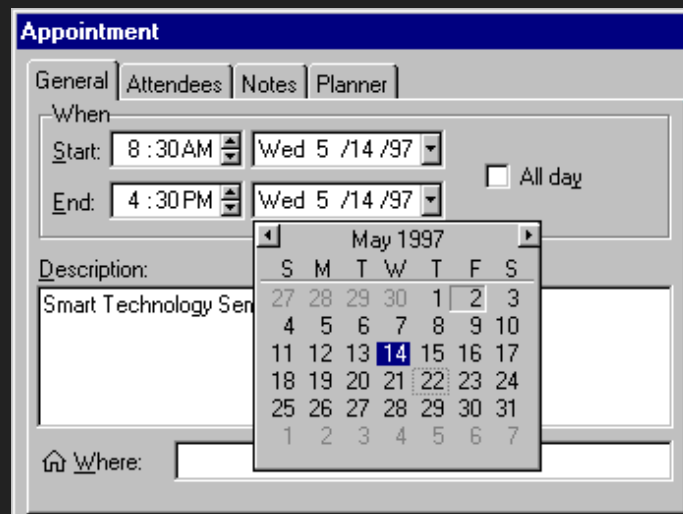


# H5: error prevention

- 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



The screenshot shows a window titled 'Form1' with a 'Date:' label. Below the label is a large empty text box. Underneath are three small input boxes for 'Month', 'Day', and 'Year'. Below these is a date picker showing 'May 22 1997'. At the bottom, there are three dropdown menus for 'Month', 'Day', and 'Year', each with a small arrow icon.



The screenshot shows an 'Appointment' dialog box with tabs for 'General', 'Attendees', 'Notes', and 'Planner'. The 'When' section has 'Start' and 'End' time and date pickers. The 'Description' field contains 'Smart Technology Sen'. A calendar widget for 'May 1997' is open, showing the date '14' selected. The 'Where' field is at the bottom.

S	M	T	W	T	F	S
27	28	29	30	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

# H5: errors we make

- **Mistakes**

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

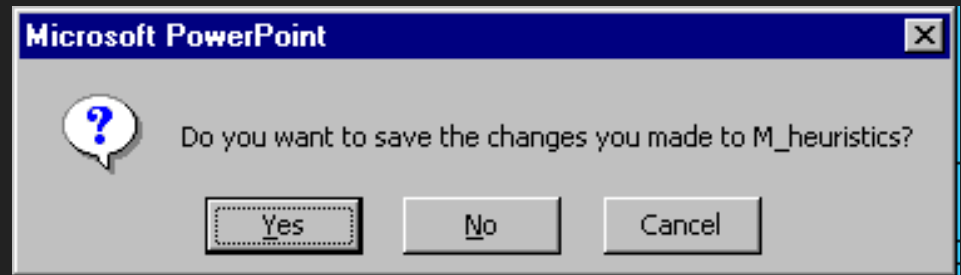
- **Slips**

- 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**

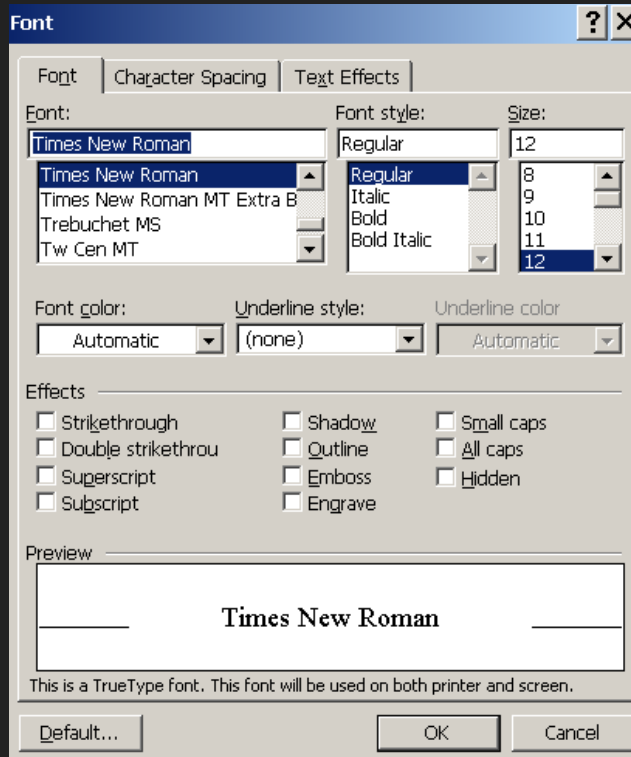
- Forgetting the goal while carrying out the action sequence  
e.G. Start going to a room and forget why by the time you get there
  - Navigating menus/dialogs, can't remember what you are looking for
  - But continue action to remember (or go back to beginning)!

- **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

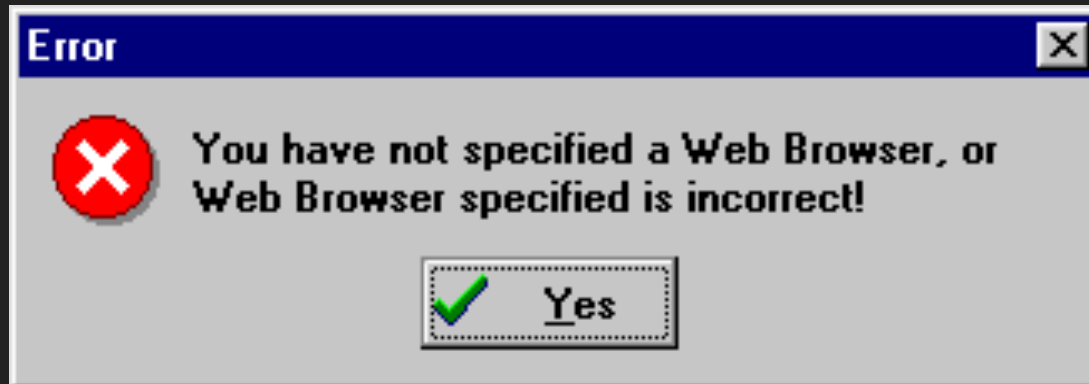
# H8: aesthetic and minimalist design

- 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.

Form Title -- (appears above URL in most browsers and is used by WWW search)		Background Color:
Q&D Software Development Order Desk		FFFBF0
Form Heading -- (appears at top of Web page in bold type)		Text Color:
Q&D Software Development Order Desk <input type="checkbox"/> Center		000080
E-Mail responses to (will not appear on)	Alternate (for mailto forms only)	Background Graphic
dversch@q-d.com		
Text to appear in Submit button	Text to appear in Reset button	<input type="radio"/> Mailto
Send Order	Clear Form	<input checked="" type="radio"/> CGI
Scrolling Status Bar Message (max length = 200 characters)		
****WebMania 1.5b with Image Map Wizard is here!!****		
<input type="button" value=" &lt;&lt; Prev Tab"/>		<input type="button" value=" Next Tab &gt;&gt;"/>

# 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

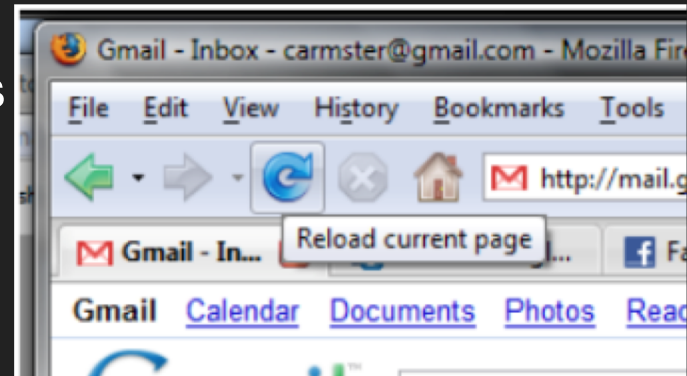
- Help is not a replacement for bad design!
- Simple systems: walk up and use; minimal instructions
- Most other systems:
  - Feature-rich
  - Some users want to become “expert” rather than “casual” users
  - Intermediate users need reminding, plus a learning path
- Many users do not read manuals
- Usually 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

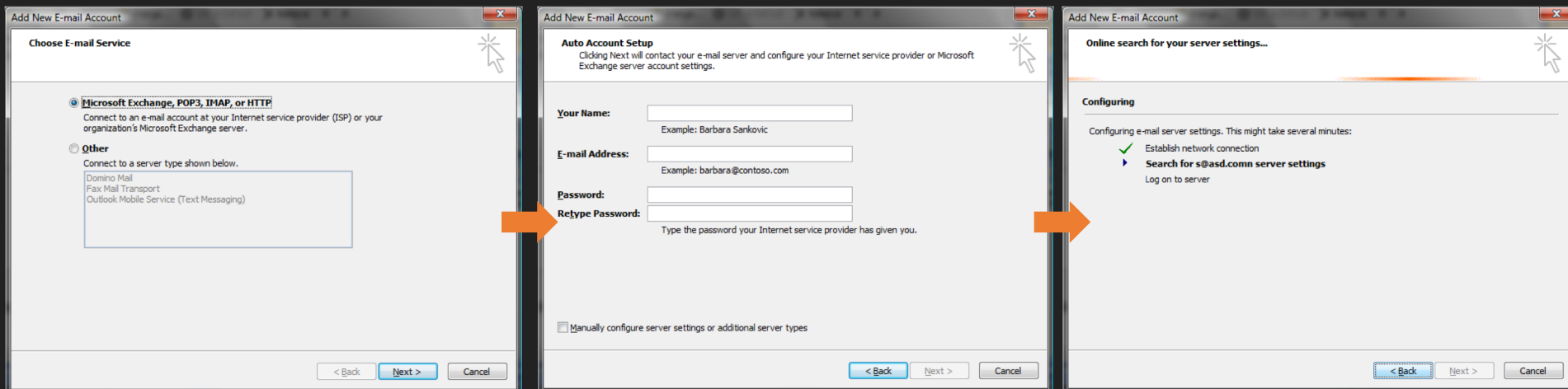
# H10: types of help (cont'd)

- 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/>