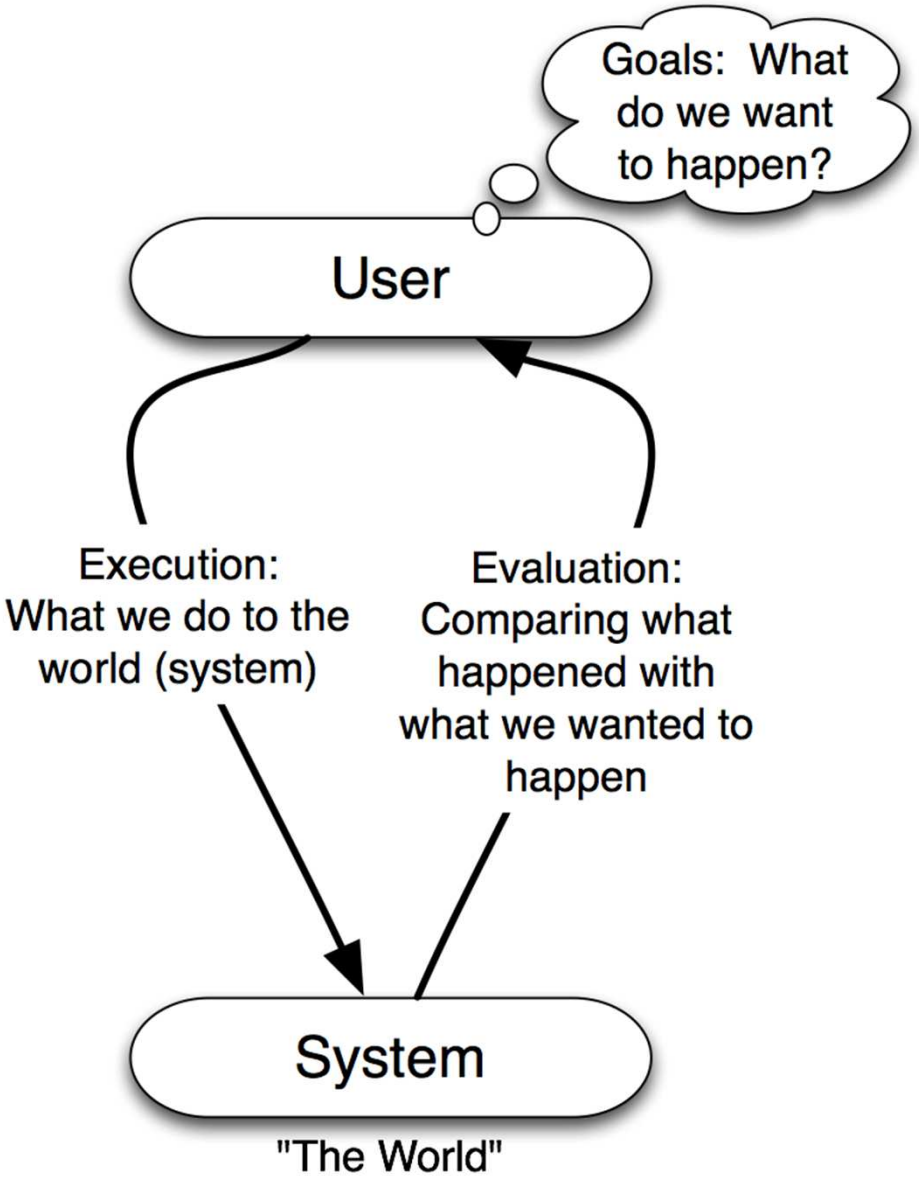
Design Guidelines

Overview

- UI design is primarily about supporting interaction
  - Models of Interaction
  - Don Norman: Examples from every day things
  - Conceptual Approaches to UI Design
    - Affordance Language
    - Paul Grice (1967), Alan Cooper (1999), Isaacs & Walendowski: Examples from butlers
    - Extending Collaborative Behaviour from *Designing from both sides of the screen* by Isaacs and Walendowski
    - *Designing Interactive Systems: People, Activities, Contexts, Technologies*, by Benyon, Turner, and Turner
  - Concrete Guidelines
    - Gnome
    - Apple's *OS X Human Interface Guidelines*

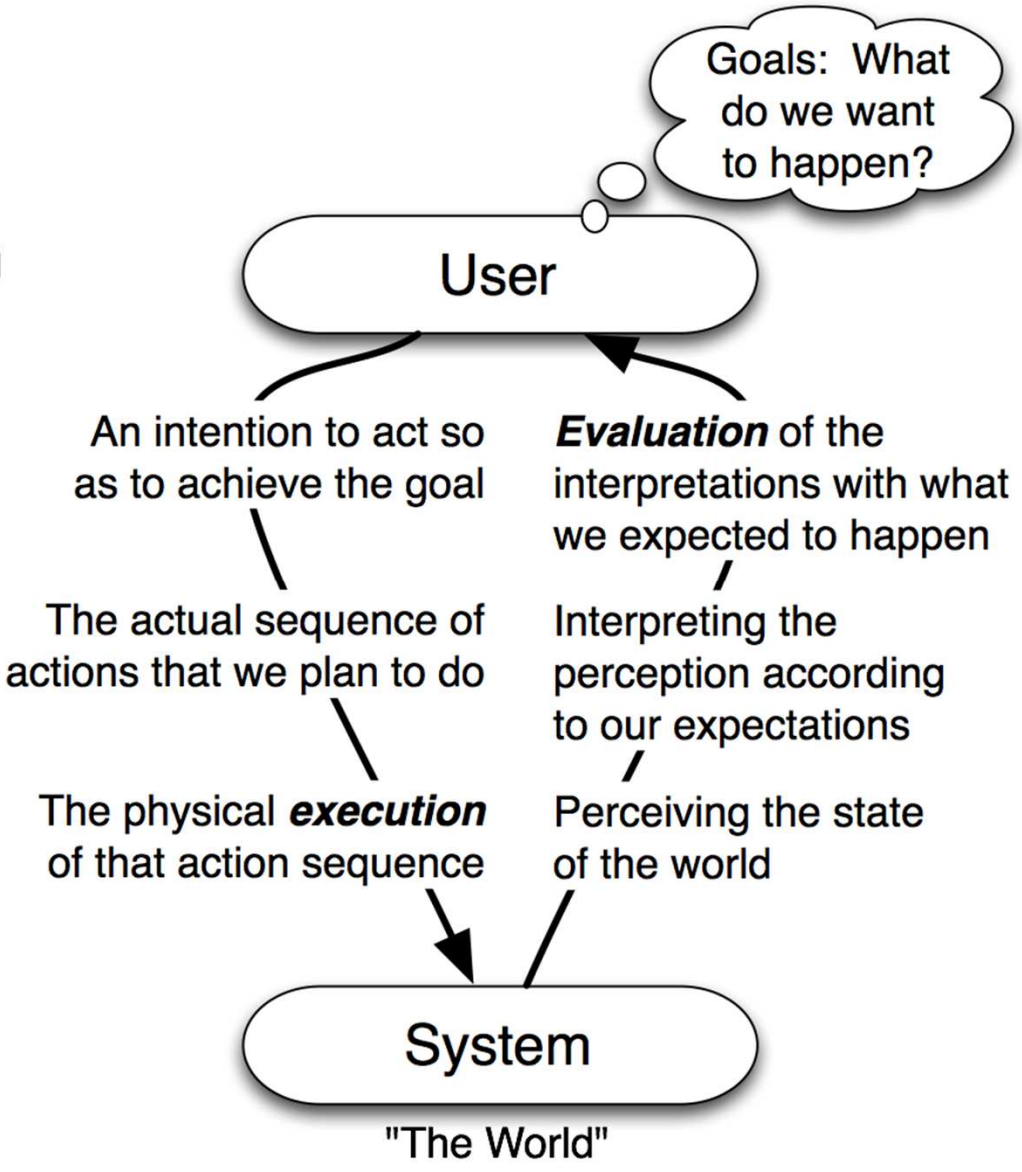
Models of Interaction

- Don Norman, *The Design of Everyday Things,* developed a general model of interaction with things (1980)



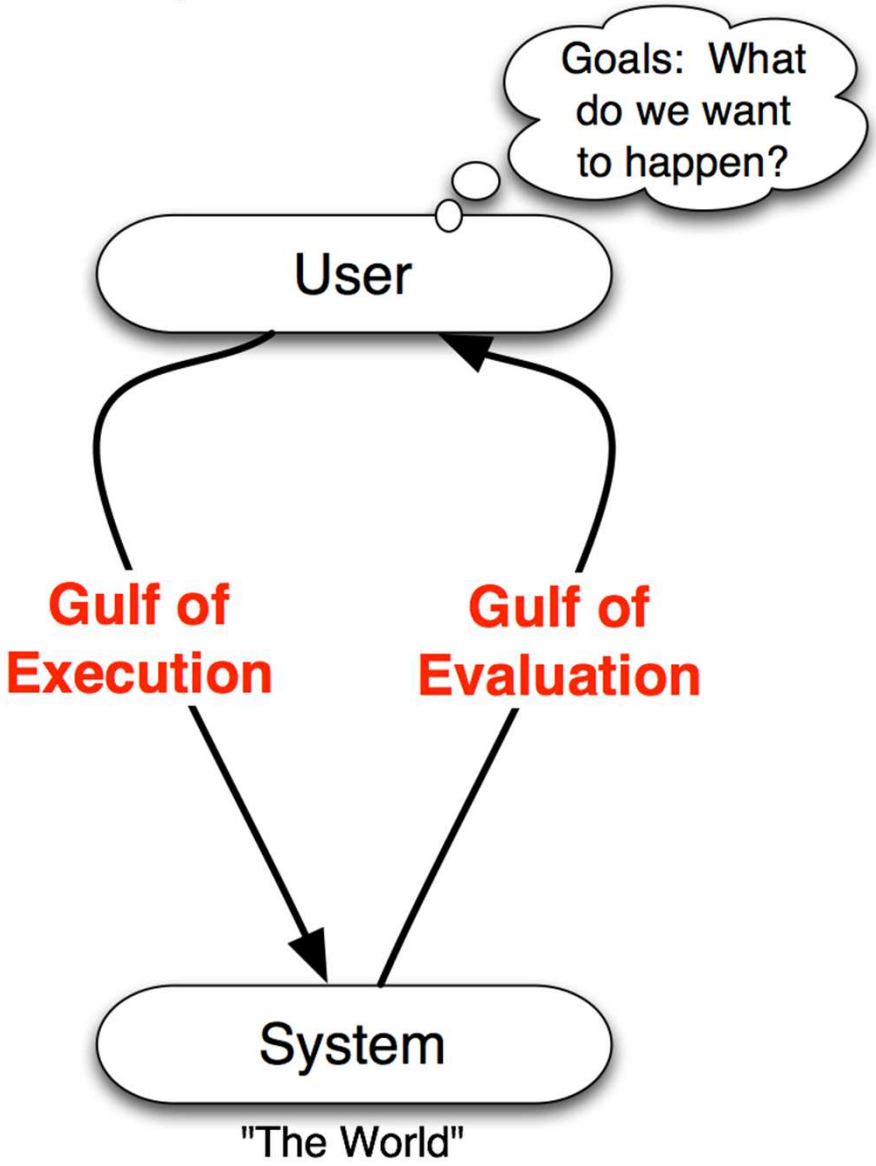
Execution and Evaluation

- Intention: a specific action taken to reach the goal
- Action Sequence: a specific list of things to do



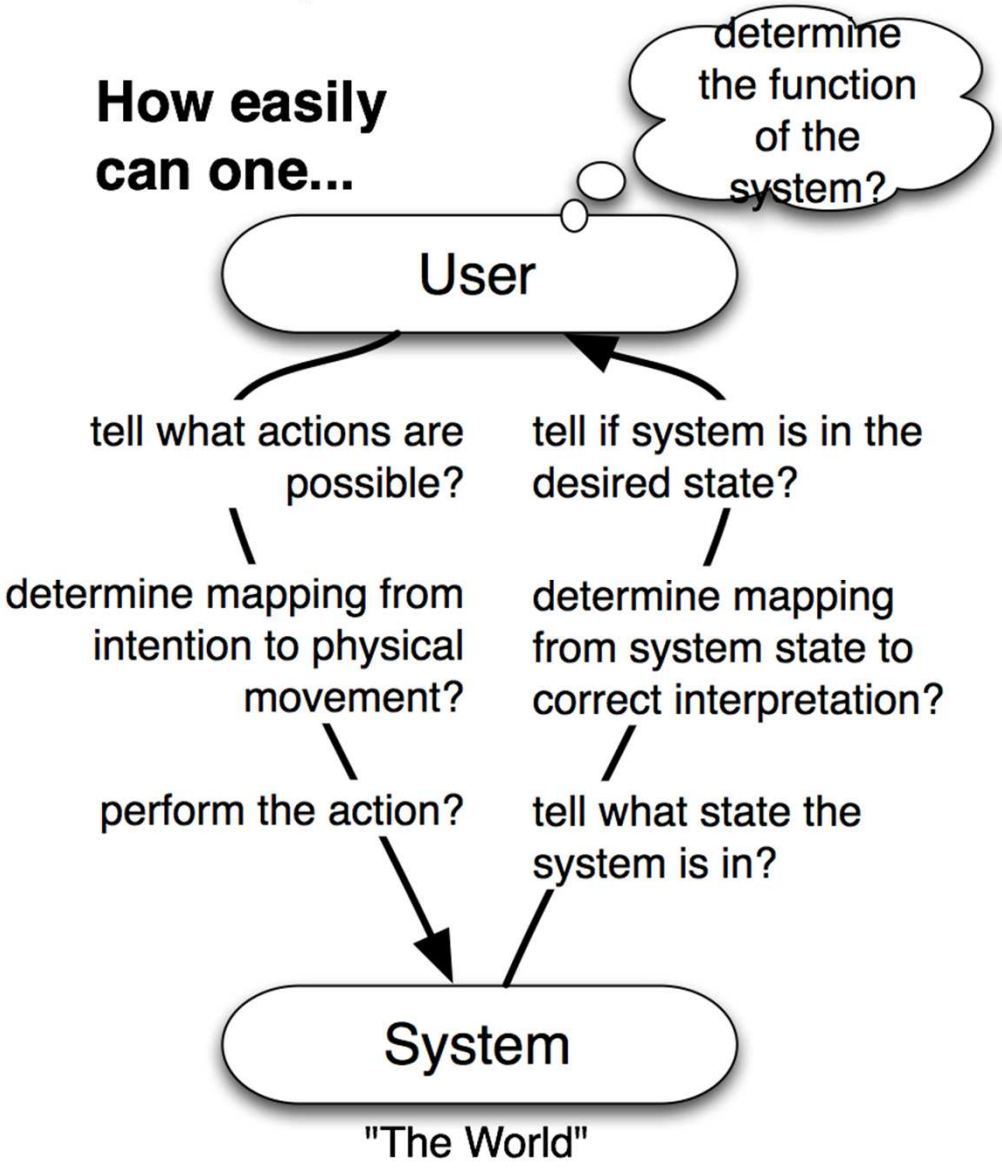
Gulfs of Execution; Evaluation

- **Gulf of Execution**: Difficulty in translating the user's intentions into actions allowed by system. Can the user carry out their intentions directly?
- **Gulf of evaluation**: Difficulty in interpreting the state of the system to determine whether the intentions have been met.



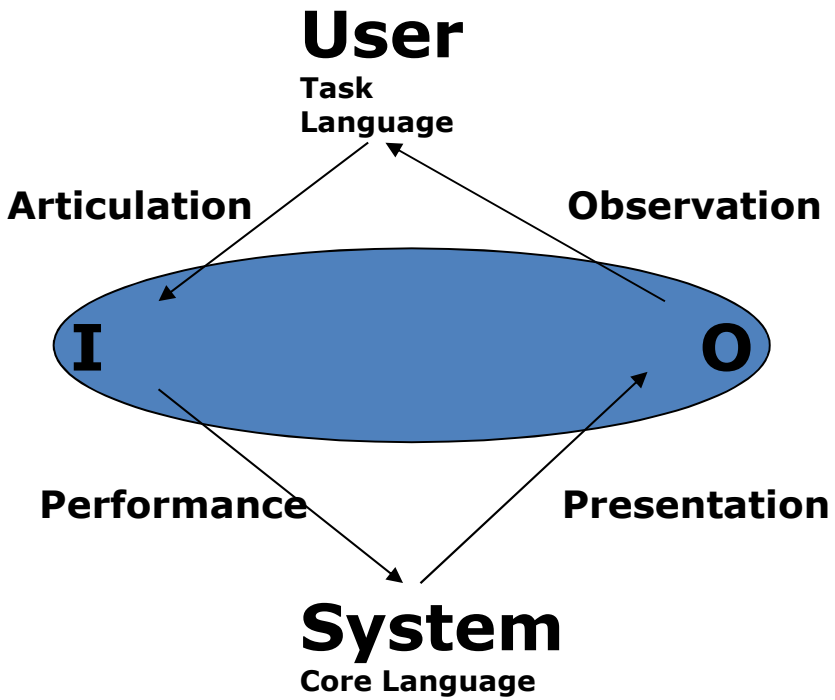
Gulfs of Execution; Evaluation

- Value:
  Provides concrete questions to ask when evaluating a system



Interaction Framework

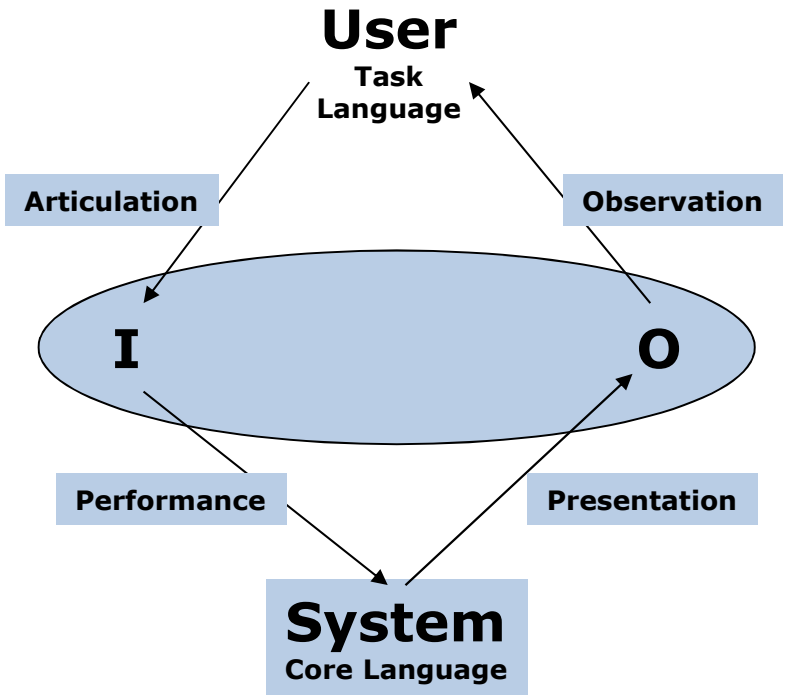
- Extends Norman's model:
  - Includes system state explicitly
- Four nodes:
  - System, User, Input and Output
  - Each node has own language:
    - System language = core language
    - User language = task language
    - Input and Output languages form the interface
      - Translates between core and task language

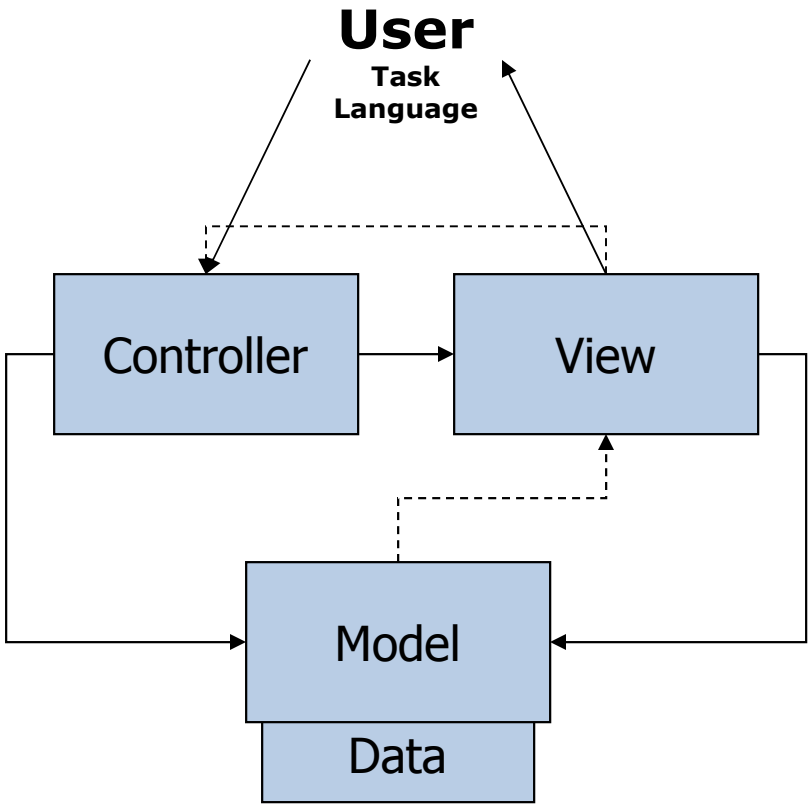


Interaction Framework

Value: Links nicely to MVC architecture

Value: Conceptualizes design of UI as translation between two languages





Models of Interaction

- High level overview of “what the UI is all about”
  - Mediates between things a user wants to do and things a system can do
  - Translates between those two world views
- UI Design translates between task language and core language
- Ultimate goal of design is to *guide use* of the system
- Norman’s book *The Design of Everyday Things* extensively discusses how design guides use

Norman: Everyday Things -- Doors

Norman: Everyday Things -- Doors

Norman: Everyday Things -- Phone

Possible Actions:

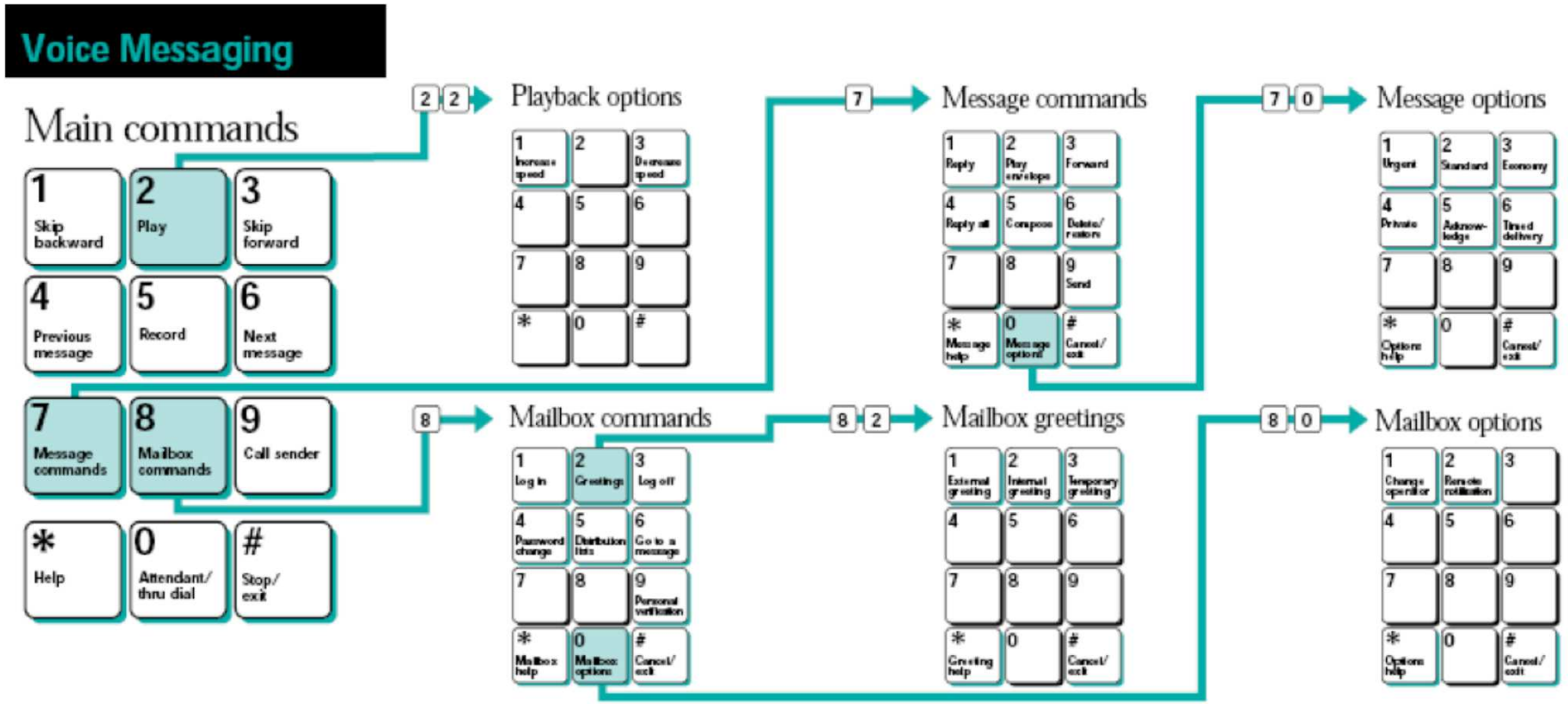
- Pick up incoming call
- Call another phone (local, long distance, extension, speed dial)
- Call operator, directory assistance, help
- Program speed dial
- Form a three-way conference call
- Transfer incoming call
- Place caller on hold
- Last number redial

Manual's instructions for redial: "
1. Press free line key.
2. Press 'Last No.' or press the line key again."

Norman: Everyday Things -- Phone

- Voice Mail:
  - Log on (from this phone, from another pone in the system, from a phone outside the system)
  - Log off
  - Change password
  - Record personal verification
  - Record external greeting (record, reply, delete)
  - Bypass greetings
- Voice Mail (cont):
  - Play messages (pause, next msg, prev. msg, continue, skip back, skip forward, delete msg, restore msg)
  - Reply/Forward/Compose/Tag message
  - Set timed delivery
  - Compose distribution list
  - Thru-dial

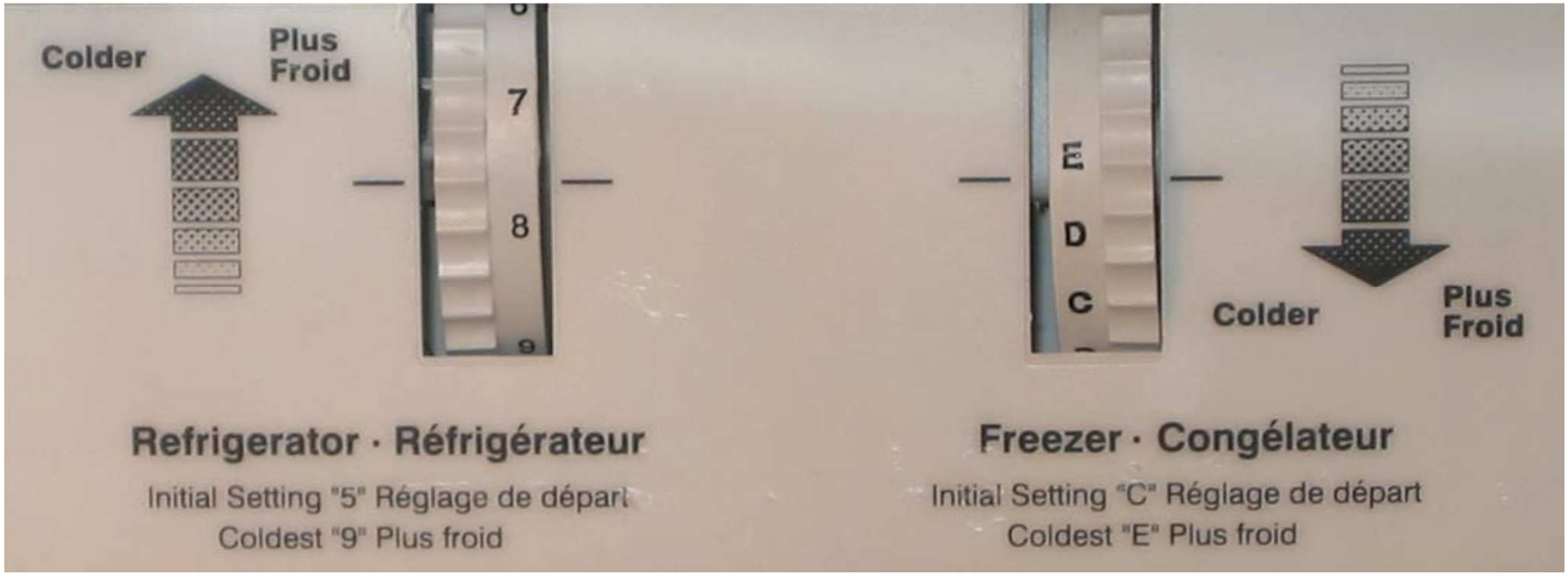
Norman: Everyday Things -- Phone



Norman: Everyday Things -- Phone

Norman: Everyday Things -- Cars

Norman: Everyday Things -- Fridge

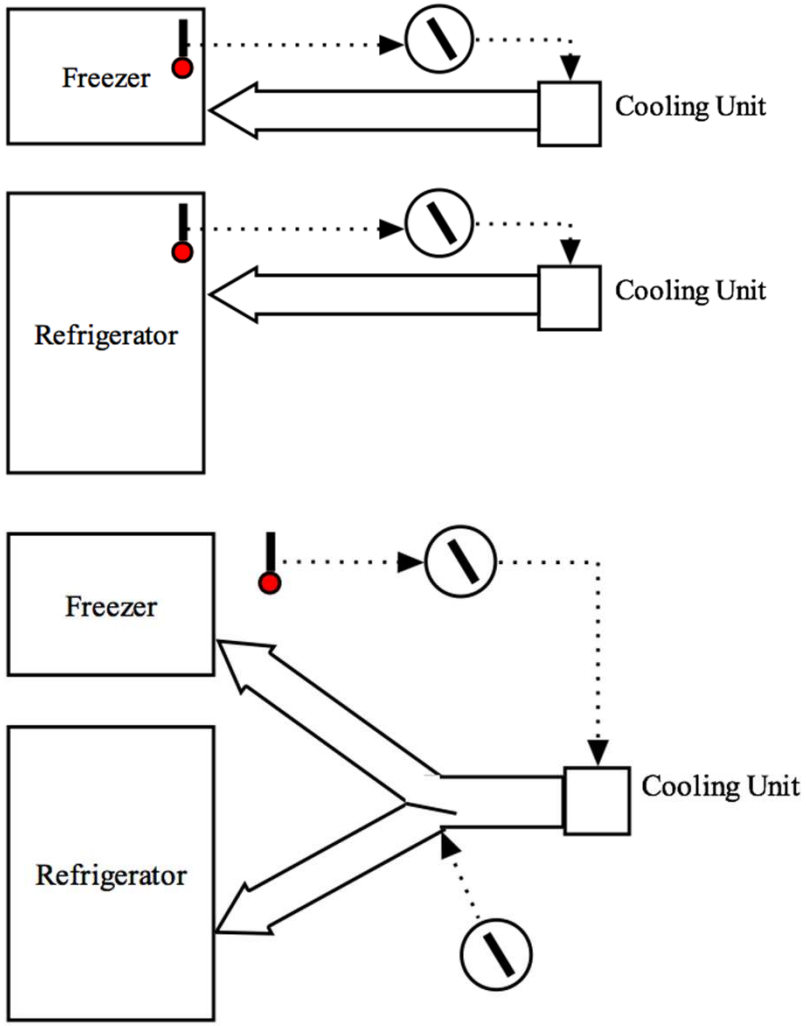


Suppose the refrigerator is at the correct temperature. The freezer is too cold. What do you do?

PS: You can't really check your work until after the refrigerator has stabilized for 24 hours and has been left closed for at least an hour or two.

Norman: Everyday Things -- Fridge

- The natural conception model is two independent controls.
- Reality:
  - Only the freezer is controlled with a thermostat (set with one of the knobs).
  - The fresh food compartment is cooled by redirecting some proportion of the cold air from the freezer.



Discussion

- What guidelines for user interfaces are suggested by these observations of everyday things?

Mental Models

- Not to be confused with models of interaction!
- A conceptual model of how things work
  - Essentially cause and effect, or hypotheses about behaviour
    - If I do this, then system does that
- Frequently, models are inaccurate or incomplete

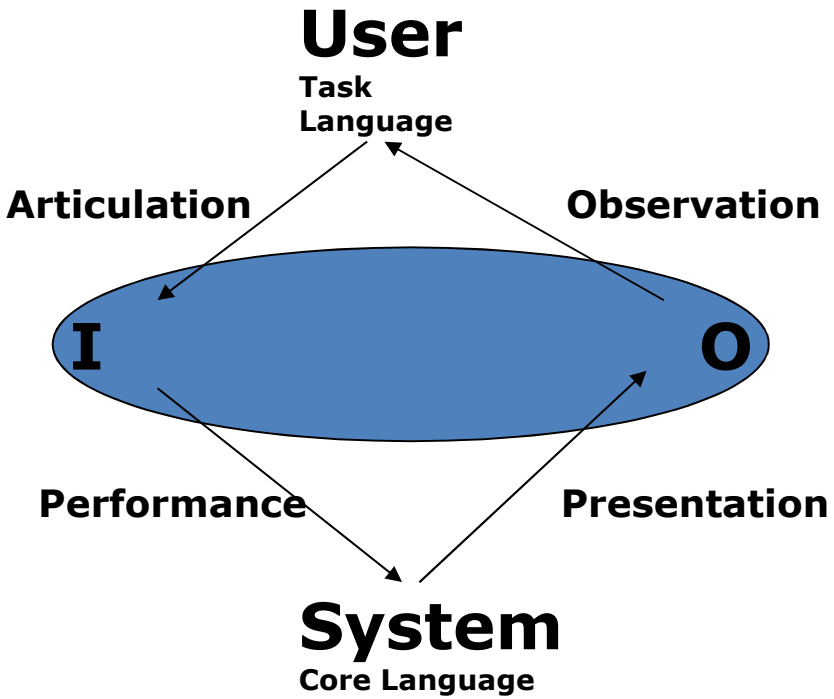
Thermostats for house and car

Mental Models

- How many ‘models’ of the system?
- Developer’s Model
  - How the programmer(s) of system believe(s) system should be used
- System image
  - The system itself
- User’s Model
  - How the user of a system believes system should be used
- Developer and User communicate via system
  - Goal is to have both images align as closely as possible

Mental Models

- User model and developer model of system are both "Mental Models" of system
  - GUI is vehicle for aligning these mental models
  - Guidelines aid UI designer in aligning models



Conceptual Approaches to UI Design

- Affordance Language
- Paul Grice (1967), Alan Cooper (1999), Isaacs & Walendowski: Examples from butlers
- Extending Collaborative Behaviour from *Designing from both sides of the screen* by Isaacs and Walendowski
- *Designing Interactive Systems: People, Activities, Contexts, Technologies*, by Benyon, Turner, and Turner

Affordance Language

- Purpose is guiding usage
  - Done by applying certain design principles to UI
  - Allows users to perceive what should be done, to map action onto display
- Essentially brings designer's model and user's model of system into alignment if done well.
- Set of principles to promote alignment of models

UI Design Principles (from Preece, Rogers, Sharp)

- Components of an *Affordance Language:*
  - Affordance
  - Mapping
  - Constraints
  - Visibility/Feedback
  - Consistency
  - Metaphor

Affordance

- Attribute of an object that allows people to know how to use it

Push

Pull

Waterloo

- Coined by Norman
  - Means “to give a clue”
  - Book: *The Design of Everyday Things*
- Norman’s current argument:
  - Should not pay too much attention during UI design
  - Objects have “real affordances”
  - Screen widgets have “perceived affordances”
  - Learned conventions

Mental Models and Affordance

- Recall thermostat examples
  - My flower version

Mental Models and Affordance

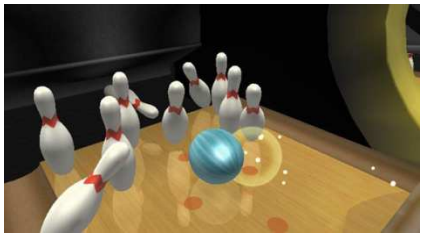
- Consider thermostat
  - What was wrong with my flower model?

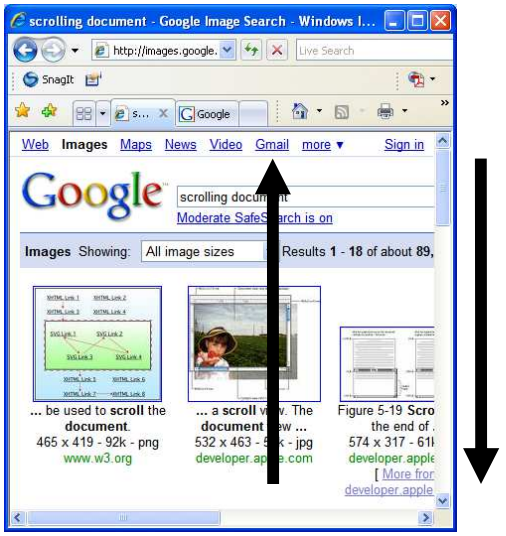
Affordance and Mental Models

- What influences our perception of affordances and the manner in which we develop mental models?
  - Individual histories
  - Cultural background
- Examples where you have developed an incorrect model or misunderstood affordance when travelling?

Mapping

- Physical actions performed on the device must be mapped onto on-screen effects
- Instruments operations must be mapped onto objects





- Recall instrumental interaction
  - Degree of integration
  - Degree of compatibility

Physical Versus Virtual

- Some things work well in physical, not in virtual

Physical

Virtual

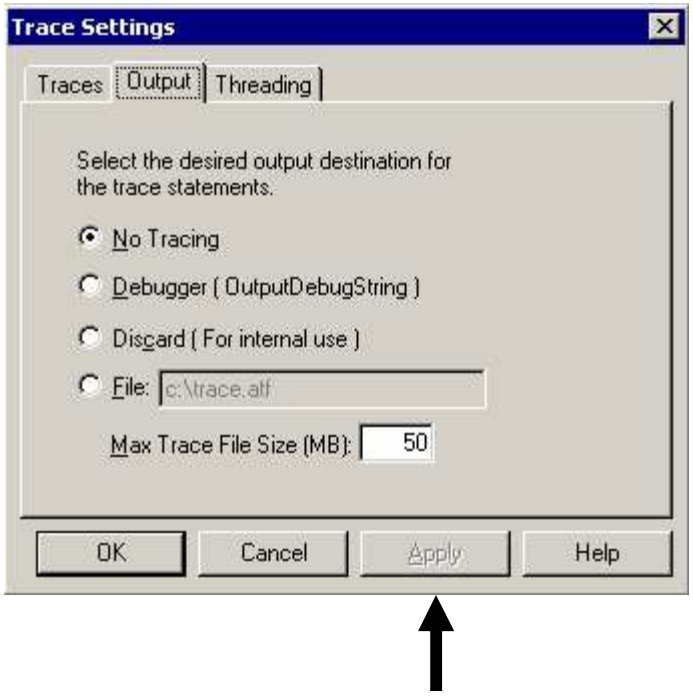
This user interface (UI) is simply hideous. Form and function have been sacrificed to looks, in the sense that it's appearance is mimicking a physical piece of equipment. Trying to adjust the parameters with the mouse is difficult and error-prone, and I am not brave enough to try using the keyboard - I don't even know how I would do that.

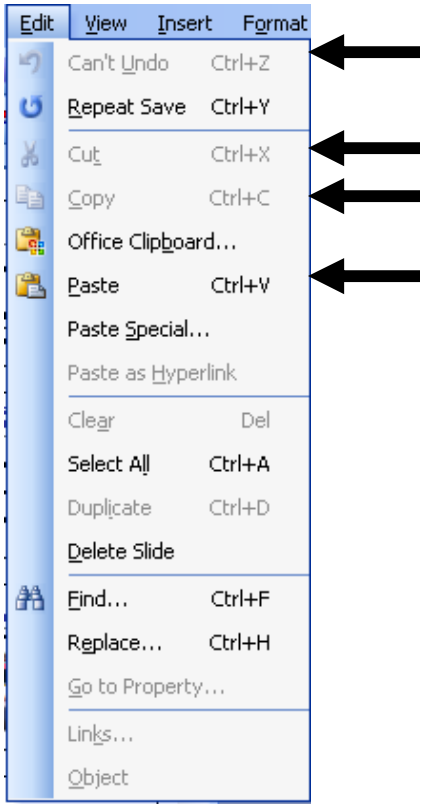
Principle: Mappings

- Mappings: "The relationship between two things, in this case between the controls and their movements and the results in the world."
  - Car:
    - Natural mapping between turning the steering wheel and turning the car.
    - Less natural for turn signals (up = left, down = right, but…)
    - Not so natural for adjusting the sound between the front and rear speakers.
  - Doors: bars/plates for pushing, handles for pulling
  - Conventions: up/clockwise for "more"
- GUIs: Components often mimic physical controls and follow same conventions and mappings.

Constraints

- Guide user by preventing certain actions, enabling others

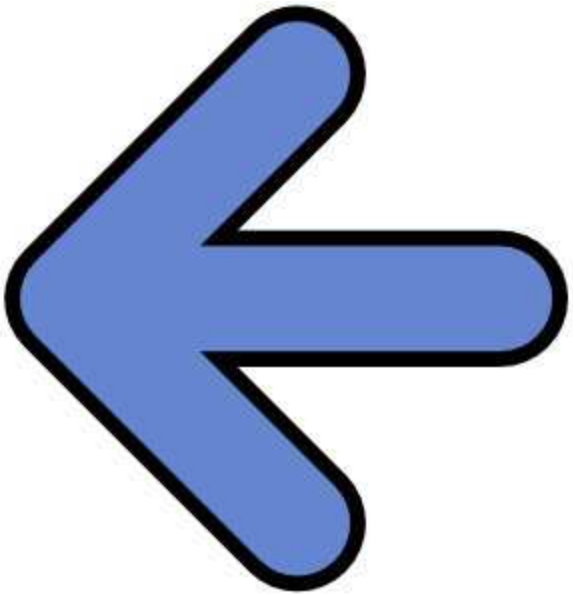




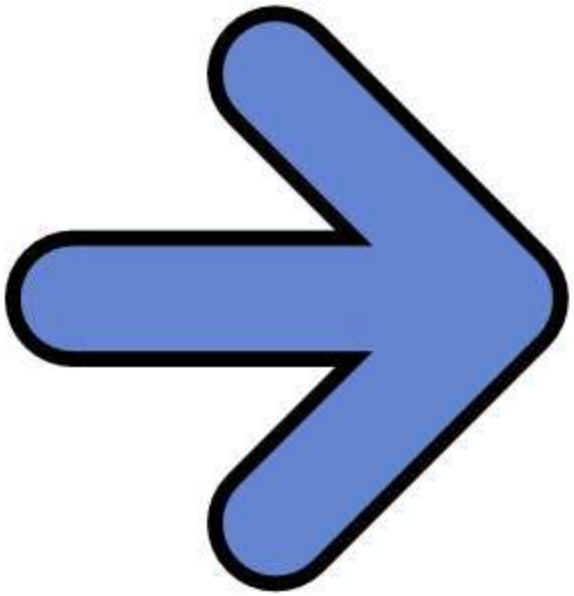
Constraints

- Physical
  - Only one way to connect
  - Disabled buttons/menus
- Logical
  - My Documents, My Pictures, etc.
- Cultural
  - Examples?

Cultural Constraints



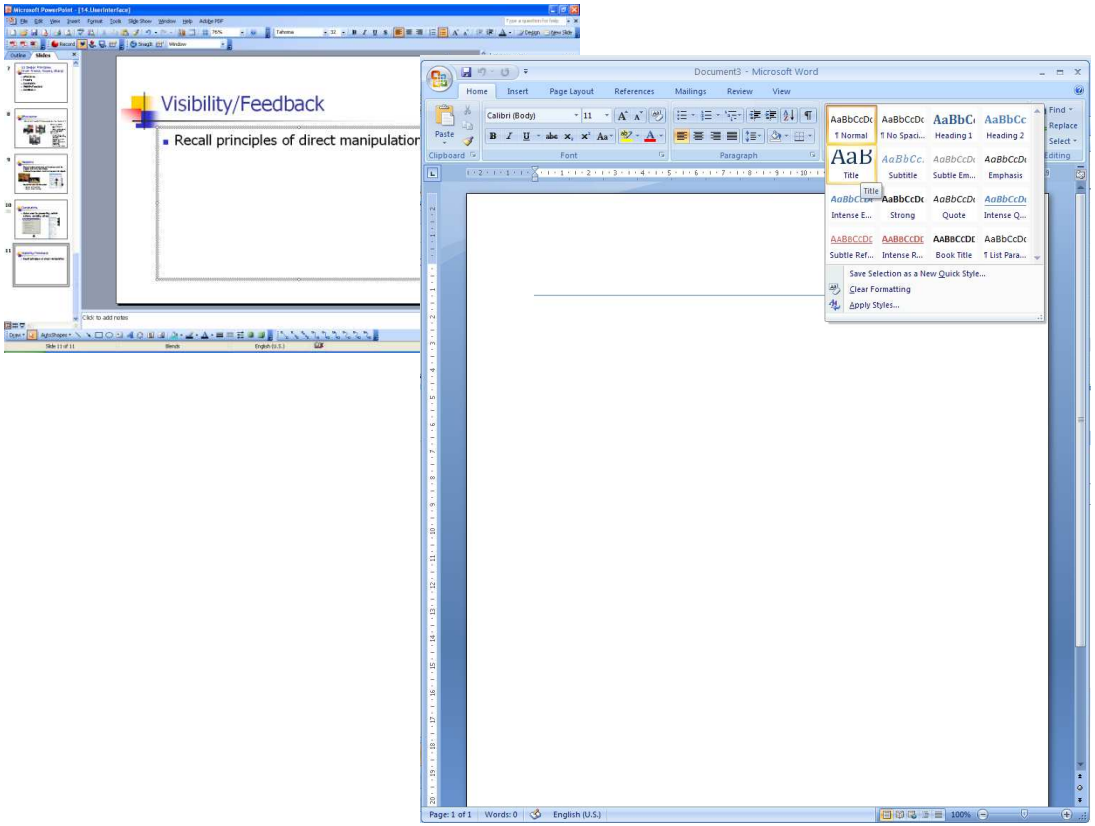
Next



Prev

Visibility/Feedback

- Recall principles of direct manipulation



- Continuous representation
- Actions on objects of interest
- Fast, incremental operations with immediate feedback
- Layered learning/self-revealing

Principle: Visibility

- Visibility: "The correct parts must be visible and they must convey the correct message."
  - Doors: Parts often gave the wrong message (pull vs. push)
  - Phones:
    - No visible indication of many available functions
    - Access to functions is arbitrary (meaningless) key sequences
  - Cars: most functions have a visible control

- GUIs: Make controls visible, either on-screen on in menus. List keyboard short-cuts in menus.

Component Feedback

- Does component effectively communicate:
  - That it is enabled/disabled?
  - That it is active/inactive?
  - Its current state?
- Does it adhere to principles of consistency and congruency across these dimensions?
- Does feedback communicate affordances?
- Can user build a suitable mental model?

UI Feedback

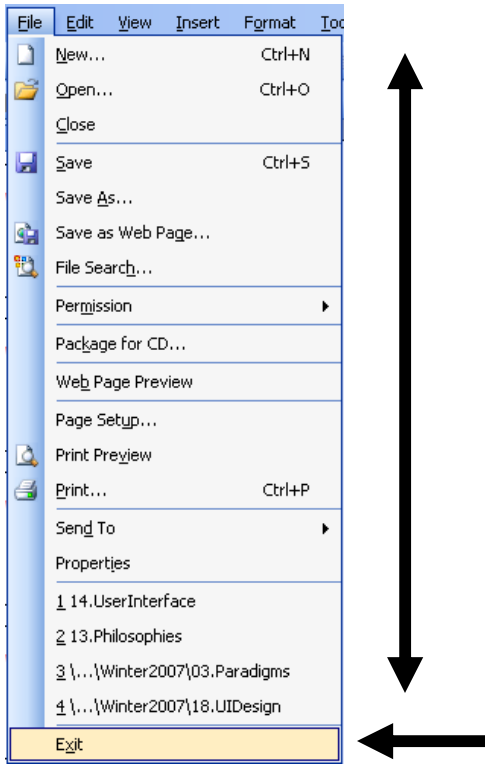
- When user acts
  - Does something happen on screen?
  - Is the user able to perceive new state of system model once action is complete?
- Examples of poor feedback?
  - Creating symbolic links in Linux
  - Sometimes “Undo” operations if they don’t correctly communicate new state.
  - Etc.

Principle: Feedback

- Feedback: "Sending back to the user information about what action has actually been done; what result has been accomplished."
  - Phones: How do you know you pressed the right sequence of buttons?
  - Car: lots of physical feedback
    - "G" force when turning/accelerating/braking.
    - Audio/visual feedback when blinkers are on.
  - Refrigerator: Feedback loop is so terribly slow.
- GUIs: Every user action should give feedback. If it results in something that can't be completed immediately, give some sort of progress indicator.

Consistency

- Allows users to leverage control from familiar onto new



Consistency

- Does component conform to rest of interface's conventions?
  - Consider both appearance and interaction
- Inconsistency can lead to
  - User frustration
  - Increased learning time
  - Errors
- Inconsistency should be carefully considered

Metaphors

- Set of unifying concepts in a GUI used to simplify interaction with a computer system
- Done by borrowing concepts from one domain (the source or vehicle) and applying them to another (the target or tenor)
- Scale can vary from system to application to UI feature
- Examples:
  - The desktop metaphor in windowing systems
  - Assembly-line metaphor for a payment system of a car manufacturer …
  - The spring metaphor in iPhone/iPad apps

Benefits of Metaphors (2)

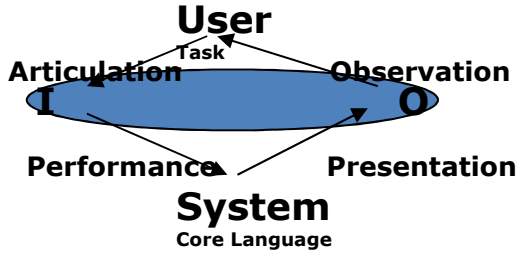
- Common language for objects
  - Window, Recycle Bin/Trash, Folders, Files
- Guide for cognitive semantics of system
  - Windows allow you to look into a house or into a document
  - Recycling allows you to reclaim storage
- Analogy to explore similarities and differences
  - Computer window has scrollbars, more similar to a repositionable viewport
  - Differences arise because characteristics of the target/tenor cause inconsistencies in the metaphor

Inconsistencies in Metaphors

- Original Mac trash
  - Delete files on computer
  - Eject disk from drive
- File system metaphor
  - Original Mac had all file systems on desktop
  - BeOS had external drives on the desktop and internal drives in a “Computer” icon
  - Windows had all file systems in a “Computer” icon

Inconsistencies in Metaphors(2)

- Inconsistencies arise because design metaphors are really **blends**
- Combine concepts that map from source with concepts that are inherent in target
  - Windows combine physical windows with the constraint of sharing screen space
  - Folders blend the concept of physical folders with location-based file storage
- So …
  - You need a good mapping
  - But you must also break that mapping to reveal needed details of target domain



Metaphor Design

- Given an idea for a metaphor, contrast features of source and target domain
- Analyze relationship between features
  - Too many features from base domain results in conceptual baggage
  - Too few features leads to confusion, poor mapping, poor metaphor
- Experiment (e.g. person-down-the-hall testing) to see if people can use metaphors to derive expectations of behaviours

Guidelines for Designing Metaphors

- Integration
  - Are metaphors coherent? Do they support a structured mapping across concepts in the metaphor?
- Unpacking
  - Can people determine why each component of the metaphor was included? Are the things chosen for inclusion cohesive?
- Topology
  - Is there a similarity in structure (abstract or concrete) between the source and target domain?
- Analysis
  - Can users use the metaphor to infer functionality?
- Visual Presentation
  - Can objects and actions be presented in a way that guides user to metaphor's concepts?

Congruence of models

- Is the component congruent with our understanding of the world?
- Can we transfer mental models from the real-world or other phenomena to our understanding of component and its usage?

Principle: Congruence

- "A good conceptual model allows us to predict the effects of our actions. Without a good model we operate by rote, blindly; we do operations as we were told to do them; we can't fully appreciate why, what effects to expect, or what to do if things go wrong. As long as things work properly we can manage. When things go wrong, however, or when we come upon a novel situation, then we need a deeper understanding, a good model."

Another Approach: Butlers

- Always available.
- Prepared to fulfil requests, with few questions and no complaints.
- If there is a problem, he finds a way to fix it or work around it without bother his employer.
- He rarely interrupts to suggest ways he can
  Instead, he pays attention to what his emp
  the past so he can better anticipate what she will want in the future. Still, doesn't go overboard.
- He makes a special effort to be courteous and respectful, even when his employer asks for things he can't do.

Respecting User Effort

- One of the best take aways of the Butler material is to respect the user's effort
  - Respect their physical effort
  - Respect their mental effort

Respect Physical Effort

- “Think of physical effort as a measure of the effort required to use your technology…. Each click represents another step in the process, and after a short while, those steps add up so that using your technology feels burdensome rather than pleasant.”

  – Treat clicks as sacred
  – Remember where they put things
  – Remember what they told you
  – Stick with a mode

Respect Mental Effort

- "Just as you should reduce your customers' physical effort, you should also reduce their mental effort. You want to leave them as much mental energy as possible so they can flow with their task and forget about the technology entirely."
  - Use visual elements sparingly
  - Give feedback; show signs of progress
  - Follow conventions (even if they aren't your ideal design)
  - Make common tasks visible; hide infrequent tasks
  - Keep preferences to a minimum; use smart defaults
  - Look for "widgetless features"

Extending Collaborative Behavior

- For decades, social scientists have been studying collaborative behavior, particularly as it relates to conversation. We use some of their observations in drawing up our own rules about how technology should cooperate with people. There are two types of politeness: negative politeness and positive politeness.
  - To be negatively polite one should do no harm; don't ask too much of people, don't impose, don't offend...
  - If you want to delight your users, you can design it to be 'positively polite' by actively cooperating...
    - *Designing from both sides of the screen, p.12*

Cooperative Principles for Tech

- Don’t Impose (negative politeness)
  - Respect the user’s physical effort
  - Respect the user’s mental effort
- Be Helpful (positive politeness)
  - Offer sufficient information early and in context; prevent errors
  - Solve problems; don’t complain or pass the buck
  - Be predictable
  - Request and offer only relevant information; don’t mislead
  - Explain in plain language

An Overview List

- High-Level Statement:
  Helping people access, learn, and remember the system while giving them the sense of being in control, knowing what to do and how to do it safely and securely in a way that suits them.
  - From *Designing Interactive Systems: People, Activities, Contexts, Technologies*, by Benyon, Turner, and Turner, Addison-Wesley, 2005, p. 65-66.

- Helping people access, learn, and remember the system…
- **Visibility**: Try to ensure that things are visible so that people can see what functions are available and what the system is currently doing.
- **Consistency**: Be consistent in the use of design features and be consistent with similar systems and standard ways of working.
- **Familiarity**: Use language and symbols that the intended audience will be familiar with.
- **Affordance**: Design things so it is clear what they are for.

- Giving them the sense of being in control, knowing what to do and how to do it……
- **Navigation**: Provide support to enable people to move around the parts of the system: maps, directional signs and information signs.
- **Control**: Make it clear who or what is in control and allow people to take control. Control is enhanced if there is a clear, logical mapping between controls and the effect that they have.
- **Feedback**: Rapidly feed back information from the system to people so that they know what effect their actions have had.

- Safely and securely……
- **Recovery**: Enable recovery from actions, particularly mistakes and errors, quickly and effectively.
- **Constraints**: Provide constraints so that people do not try to do things that are inappropriate.

- In a way that suits them…
- **Flexibility**: Allow multiple ways of doing things so as to accommodate users with different levels of experiences and interest in the systems. Provide people with the opportunity to change the way things look or behave so that they can personalize the system.
- **Style**: Designs should be stylish and attractive.
- **Conviviality**: Interactive systems should be polite, friendly, and generally pleasant.

Example GNOME Guidelines

- GNOME Human Interface Guidelines (HIG)
  - http://library.gnome.org/devel/hig-book/stable/
  - “This document tells you how to create applications that look right, behave properly, and fit into the GNOME user interface as a whole.”
- Menu Examples:
  - “Label menu items with verbs for commands and adjectives for settings”
  - “Make a menu item insensitive when its command is unavailable”
  - Provide an access key for every menu item.”

Example Apple Guidelines

- Apple OS X Human Interface Guidelines
  - http://developer.apple.com/library/mac/#documentation/UserExperience/Conceptual/AppleHIGuidelines/Intro/Intro.html
- Menu Examples:
  - "Use menu titles that accurately represent the items in the menu."
  - "Make menu titles as short as possible without sacrificing clarity."
  - "Avoid using an icon for a menu title."
  - "Ensure that a menu's title is undimmed even when all of the menu's commands are unavailable."

What Guidelines Provide

- Explicit, well-defined rules to follow
  - Acceptable font sizes and styles
  - Minimum spacings required between controls, borders of panels
  - Color schemes to avoid for sake of color blind users
  - *In many cases, these rules can be automatically checked by compliance software*
- Higher-level principles
  - Consistency
  - Use of metaphors
  - WYSIWYG…
  - These principles typically *cannot* be automatically checked

Next Steps

- Guidelines only get you so far
- Need to test your app with real users -- as early and often as possible.