Usability

Based on material by Michael Ernst, UW and MIT

Announcements

- CHECK YOUR GRADES
  - Quiz 1-9, HW 1-6, Exam 1-2 now all in LMS!
  - Feedback on Homework in Homework Server
    - HW7 grades released. Re-grade points will be applied later as re-submit of HW7 is due December 11th
- HW9 due December 11th
  - A GUI Interface for your path finding algorithm
- Quiz 10 at the end of class

Revisit Visitor Pattern, Again

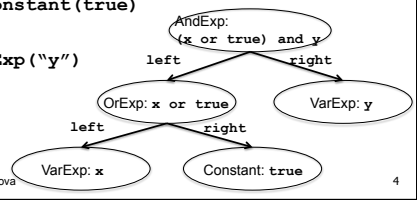
- Common questions/mistakes
- How do I "start" the Visitor?
- How to I make the Visitor hold Context?
- Can I change signature of accept or visit?
- Use of Interpreter methods in Visitors
  - You shouldn't call evaluate/print on Expressions
  - accept methods are the same for all Visitors
  - Functionality in visit methods, state stored in Visitors

Visitor's Task is to Traverse Hierarchical Structure

- Expression `(x or true) and y`

```
new AndExp(
    new OrExp(
      new VarExp("x"),
      new Constant(true)
    ),
    new VarExp("y")
)
```

We have a hierarchical structure: AndExp is top, OrExp and VarExp are below in the hierarchy, etc.



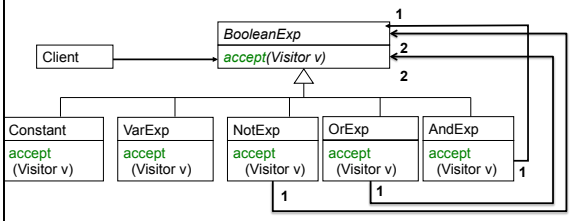
Starting the Visitor

```
BooleanExp myExp =
  new AndExp(
    new OrExp(new VarExp("x"),new VarExp("y")),
    new VarExp("z")
  );
CounterVisitor v = new CounterVisitor();
//or EvaluateVisitor v = new EvaluateVisitor(c);
//or InorderVisitor v = new InorderVisitor();

myExp.accept(v); // starts traversal at root
```

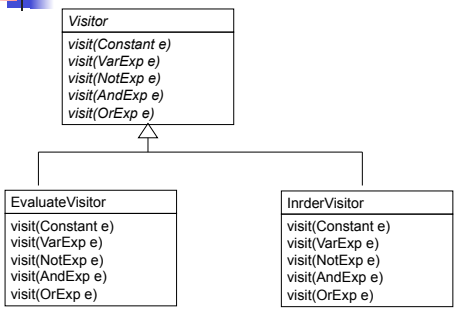

Visitor Implements Postorder Traversal of the Composite

```
class AndExp extends BooleanExp {
  public void accept(Visitor v) {
    // call accept on all children, then visit
    left.accept(v); // traverses left subexp
    right.accept(v); // traverses right subexp
    v.visit(this); // after traversal
  }
}

// accept doesn't know what kind of Visitor!
// works with all Visitors!
// No changes to the BooleanExp hierarchy
// required when adding new Visitors
```

The Visitor Pattern



The Visitor Pattern



Exercise: Write a Count Visitor that counts #nodes in a BooleanExp object

```
class VarExp extends
          BooleanExp {
 void accept(Visitor v) {
  v.visit(this);
 }
}
class AndExp extends
           BooleanExp {
 BooleanExp leftExp;
 BooleanExp rightExp;
 void accept(Visitor v) {
  leftExp.accept(v);
  rightExp.accept(v);
  v.visit(this);
 }
}
```

```
class CounterVisitor
  implements Visitor {
 int count = 0;
 void visit(VarExp e)
 {
  count++;
 }
 void visit(Constant e)
 {
  count++;
 }
 void visit(AndExp e)
 {
  count++;
 }
 …
}
```


Exercise: Write PostorderVisitor, which prints BooleanExp in postorder

```
class VarExp extends
          BooleanExp {
 void accept(Visitor v) {
  v.visit(this);
 }
}
class AndExp extends
           BooleanExp {
 BooleanExp leftExp;
 BooleanExp rightExp;
 void accept(Visitor v) {
  leftExp.accept(v);
  rightExp.accept(v);
  v.visit(this);
 }
}
```

```
class PostroderVisitor
  implements Visitor {

 void visit(VarExp e)
 {
 //print e.getString();
 }
 void visit(Constant e)
 {
 //print e.getValue();
 }
 void visit(AndExp e)
 {
 //print "AND";
 }
 …
}
```


Exercise: Write an Evaluate Visitor which evaluates a BooleanExp

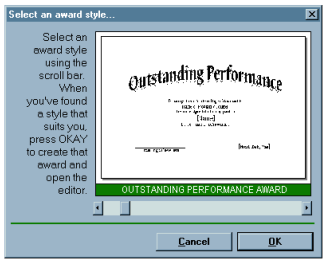
```
class VarExp extends
          BooleanExp {
 void accept(Visitor v) {
  v.visit(this);
 }
}
class AndExp extends
           BooleanExp {
 BooleanExp leftExp;
 BooleanExp rightExp;
 void accept(Visitor v) {
  leftExp.accept(v);
  rightExp.accept(v);
  v.visit(this);
 }
}
```

```
class EvaluateVisitor
  implements Visitor {
 // ??
 void visit(VarExp e)
 {
  // ??
 }
 void visit(Constant e)
 {
  // ??
 }
 void visit(AndExp e)
 {
  // ??
 }
 …
}
```


Outline of Today's Class

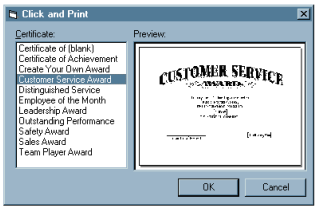
- Usability
- Iterative Design
  - Design
    - Design principles
  - Implement
    - Low-fidelity prototypes
  - Evaluate
    - User testing

User Interface Hall of Shame

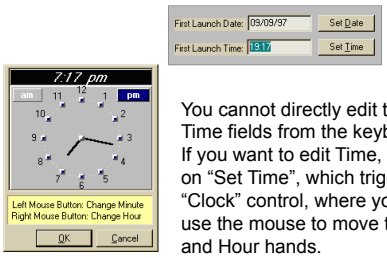


User Interface Hall of Shame

Better

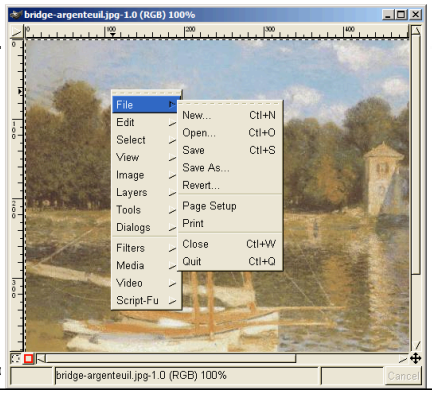


User Interface Hall of Shame

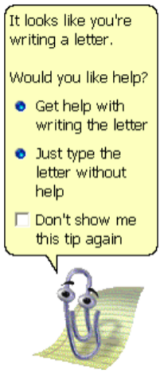


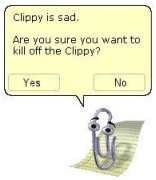
You cannot directly edit the Date and Time fields from the keyboard.
If you want to edit Time, you must click on “Set Time”, which triggers the “Clock” control, where you have to use the mouse to move the Minute and Hour hands.

Hall of Shame or Fame?



Hall of Shame or Fame?





Designing User Interfaces Is Hard

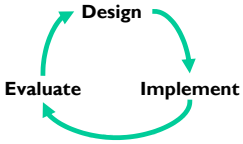
- You are not the user
- Most software engineering is about communicating with programmers
  - Who are a lot like us
- UI is about communicating with users
  - Users are NOT like us
- The user is ALWAYS right
  - Usability problems are the design's fault
  - Hard lesson to learn: if the user consistently gets stuck, this is not because the user is dumb, but because the interface is poorly designed

Designing User Interfaces Is Hard

- … unfortunately, the user is not always right
- The user cannot predict what they really want
- 1950's experiment with telephone handsets
  - Users thought weight was fine
  - Actually, they really wanted half the weight
- # of results displayed for a Google search query
  - Users say they want 30
  - Actually, they really wanted 10

Iterative Design

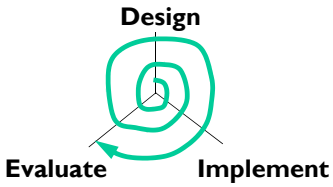
- UI development is an iterative process



- Iterations can be costly
  - If the design turns out to be bad, you many have to throw away most of your code

Spiral Model

- Use throw-away prototypes and cheap evaluation for early iterations



Usability

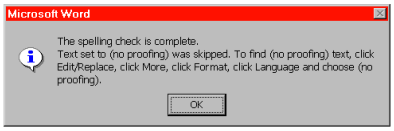
- Usability: how well users can use the system's functionality
- Dimensions of usability
  - Learnability: is it easy to learn?
  - Efficiency: once learned, is it fast to use?
  - Safety: are errors few and recoverable?
  - Memorability: is it easy to remember what you learned?
  - Satisfaction: is it enjoyable to use?

Usability Dimensions

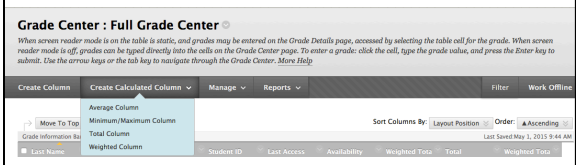
- Learnability
- Efficiency
- Safety
- Simplicity (not a usability dimension)
- Different dimensions vary in importance
  - Depends on the user
  - Depends on the task
- Usability is only one aspect of the system

Learnability
IBM's Real CD

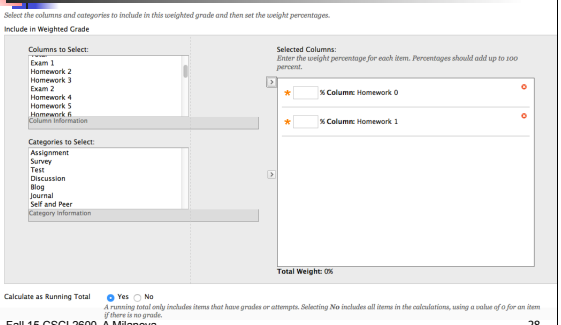
Learnability



Learnability
LMS Create Weighted Column

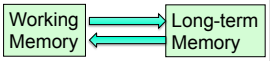


Learnability
LMS Create Weighted Column



Facts About Memory & Learning

- Working memory
  - Small: 7 ± 2 "chunks"
  - Short-lived: gone in ~10 seconds
  - **Maintenance rehearsal** is required to keep it from decaying but costs attention
- Long-term memory
  - Practically infinite in size and duration
  - **Elaborative rehearsal** transfer chunks to long-term memory

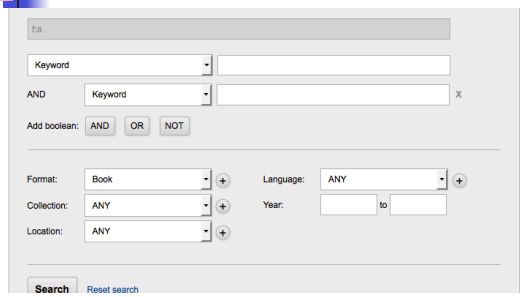


Design Principles for Learnability

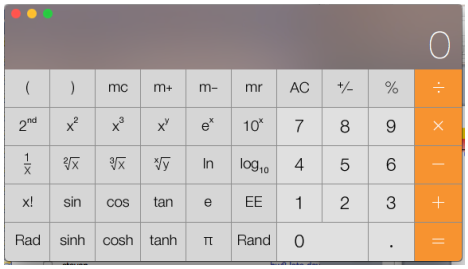
- Consistency
  - Similar things look similar, different things different
  - Terminology, location, …
  - Internal, external, metaphorical design
- Use common, simple words, not tech jargon!

- Recognition, not recall
  - Labeled buttons are better than commands
  - Combo boxes are better than text boxes

Tech Jargon
UHLS Catalog Advanced Search



Visibility



Facts About Human Perception

- **Perceptual fusion**: stimuli ~100ms apart appear fused to our perceptual system
  - 10 frames/sec is enough to perceive a moving picture
  - Computer response < 100ms feels instantaneous
- **Color blindness**: many users (~8% of all males) can't distinguish red from green

Google — normal vision

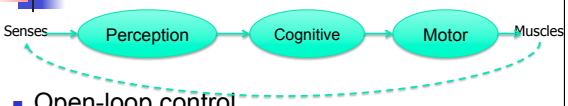
Google — red-green deficient

Design Principles for Visibility

- Make system state visible: keep the user informed about what's going on
  - Mouse cursor, selection highlight, status bar
- Give prompt feedback
  - Response time rules-of-thumb:

< 0.1 sec	seems instantaneous
0.1 – 1 sec	user notices
1 - 5 sec	display busy cursor
> 5 sec	display progress bar

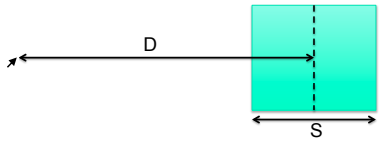
Facts About Motor Processing



- Open-loop control
  - Motor processor runs by itself
  - Cycle time is ~ 70ms
- Closed-loop control
  - Muscle movements are perceived and compared with desired result
  - Cycle time is ~ 140ms

Pointing Tasks: Fitts's Law

- How long does it take to move your hand to a target of size S at distance D away?



- E.g. moving mouse to target on screen

Fitts's Law

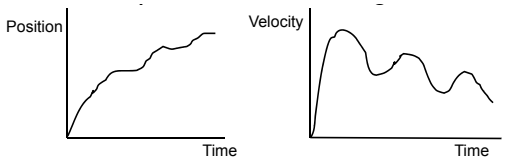
Reaction time | Movement time

- T = RT + MT = a + b log (D/S)

- log (D/S) is the index of difficulty of the pointing task

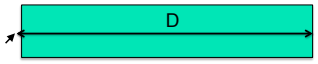
Derivation of Fitts's Law

- Moving your hand is closed-loop control
- Each cycle covers remaining distance d with error εd
- After two cycles, within $\varepsilon^2 D$ of target



Path Steering Tasks: Steering Law

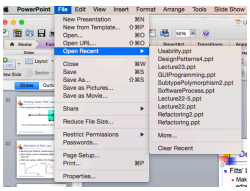
- Fitts's Law applies only if path to target is **unconstrained**
- But the task is much harder if path is constraint to a tunnel



- Steering Law: T = RT + MT = a + b (D/S)

Design Principles for Efficiency

- Fitts's Law and Steering Law
  - Make important targets big, nearby, or at screen edges
  - Avoid steering tasks!
- Provide shortcuts
  - Keyboard accelerators
  - Styles
  - Bookmarks
  - History



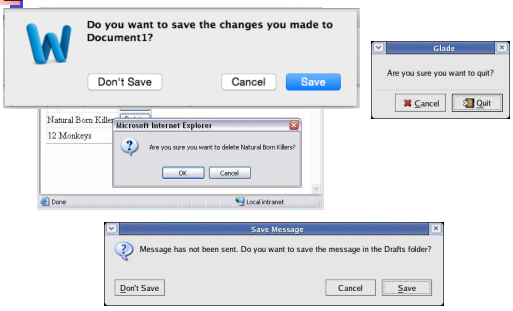
Usability Dimensions

- Learnability
- Efficiency
- Safety
- Simplicity

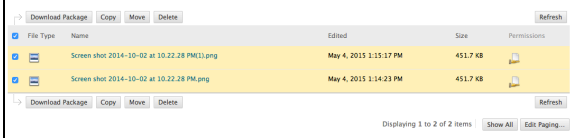
Mode Errors

- Modes: states in which actions have different meanings
  - E.g., vi's insert mode vs. command mode
- Avoiding mode errors
  - Eliminate modes entirely
  - Visibility of mode
  - Disjoint action sets in different modes

Confirmation Dialogs

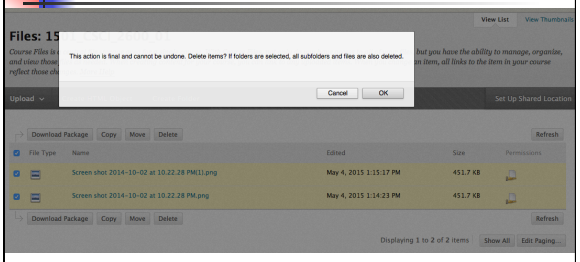


Confirmation Dialogs: Deleting files in the LMS file system



I selected the files and clicked Delete

Confirmation Dialogs: Deleting files in the LMS file system



Not done! Confirmation dialog pops up. Clicked OK

Confirmation Dialogs: Deleting files in the LMS file system

Still not done! Clicked Submit and finally done! Another issue with file system: behavior inconsistent with other systems.

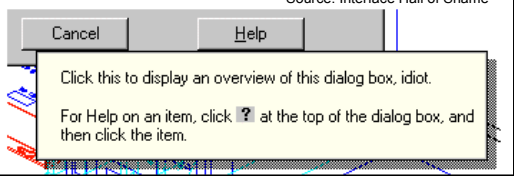
Design Principles for Error Handling (Safety)

- Use confirmation dialogs sparingly
- Prevent errors as much as possible
  - Selection is better than typing
  - Avoid mode errors
  - Disable illegal commands
  - Separate risky command from common ones
- Support Undo

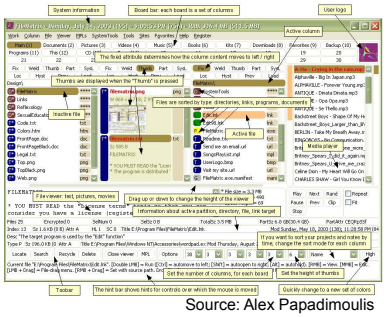
Design Principles for Error Handling (Safety)

- Good error messages
  - Precise
  - Speak the user's language
  - Constructive help
  - Be polite

Source: Interface Hall of Shame



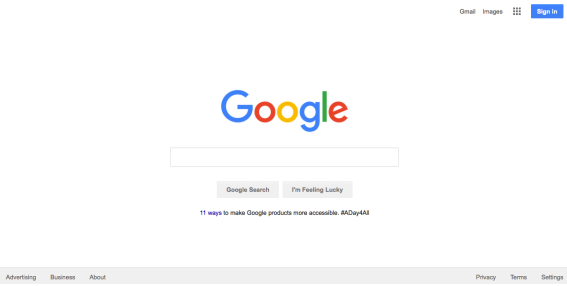
Simplicity



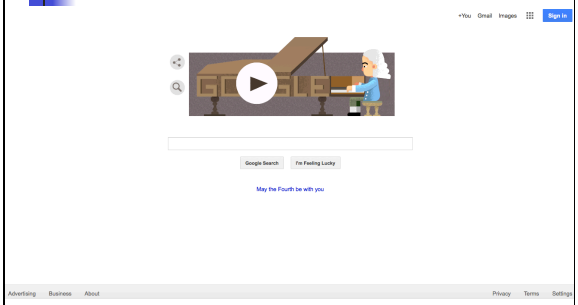
Source: Alex Papadimoulis

Simplicity, Google Back in 2003

Simplicity, Google Now



Simplicity, Google Now



Design Principles for Simplicity

- "Less is More!"
  - Omit extraneous information, graphics & features
- Good graphic design
  - Few well-chosen colors and fonts
  - Group with whitespace

- Use concise language
  - Choose labels carefully

Document Your System

- Write the user manual
  - Program and UI metaphors
  - Key functionality
  - Do not include: exhaustive list of all menus
- What is hard to do?
- Who is your target audience?
  - Power users need a manual
  - Casual users might not
- Piecemeal online help is no substitute

Outline of Today's Class

- Usability
- Iterative Design
  - Design
    - Design principles
  - Implement
    - Low-fidelity prototypes
  - Evaluate
    - User testing

Low-fidelity Prototype

- Paper is a very fast and effective prototyping tool
  - Sketch windows, menus, dialogs, widgets
  - Crank out lots of designs and evaluate them
- Hand-sketching is OK --- even preferable
  - Focus on behavior & interactions, not fonts & colors
  - Similar to design of your ADTs and classes
- Paper prototypes can even be executed!
  - Use pieces to represent windows, dialogs, menus
  - Simulate computer's responses by moving pieces around and writing on them

User Testing

- Start with a prototype
- Write up a few representative tasks
  - Short but non-trivial
    - E.g., "add this meeting to calendar",
    - E.g., "type this letter and print it"
- Find a few representative users
  - 3 is often enough to find obvious problems
- Watch them do tasks with the prototype

How to Watch Users

- Brief the user first
  - "I'm testing the system, not testing you"
  - "If you have trouble, it's the system's fault"
  - "Feel free to quit at any time"
  - Ethical issues: informed consent
- Ask user to think aloud
- Be quiet!
  - Don't help, don't explain, don't point out mistakes
  - Two exceptions: prod user to think aloud, and move on to the next task when stuck
- Take lots of notes

Watch for Critical Incidents

- Critical incidents: events that strongly affect task performance or satisfaction
- Usually negative
  - Errors
  - Repeated attempts
  - Curses
- Can also be positive
  - "Cool!"
  - "Oh, now I see."

Summary

- You are not the user
- Keep human capabilities and design principles in mind
- Iterate over your design
- Write documentation
- Make cheap, throw-away prototypes
- Evaluate them with users