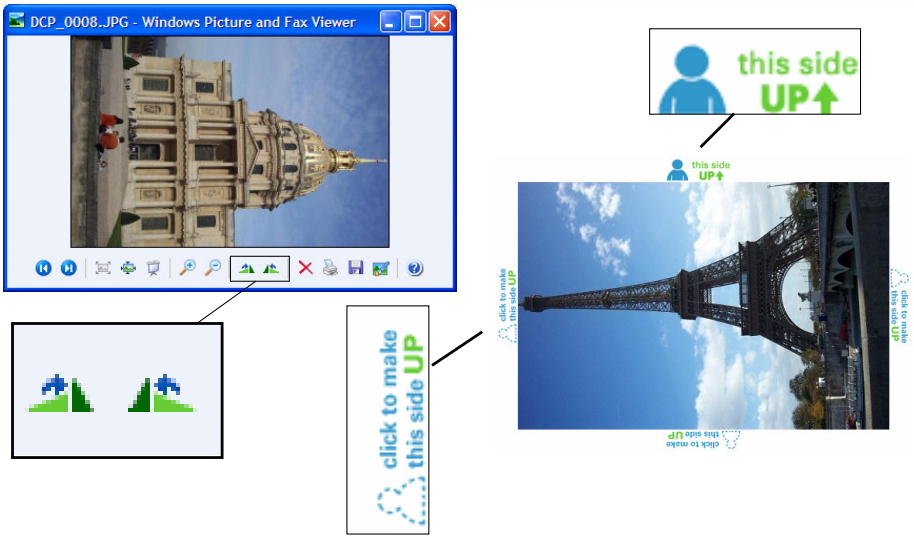
Lecture 14:
Heuristic Evaluation

UI Hall of Fame or Shame?



Fall 2004 6.831 UI Design and Implementation 2

For today's UI Hall of Fame and Shame, we'll focus on the Rotate commands in photo browsers and drawing editors. These commands rotate an image by 90 degree increments, either clockwise or counterclockwise.

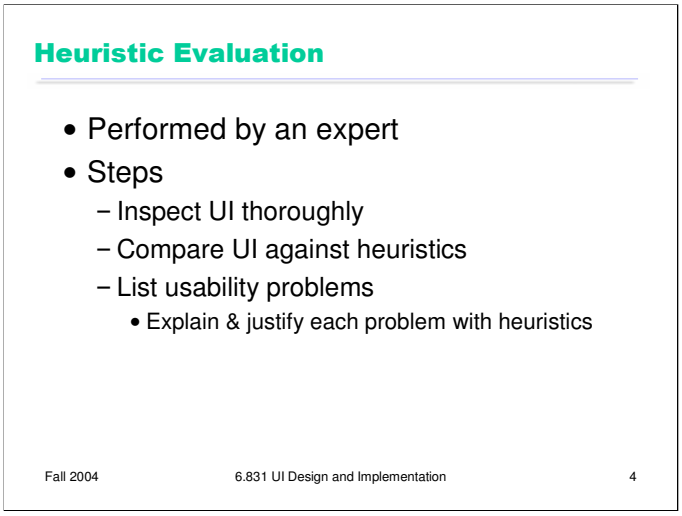
In the Windows XP Image Viewer, the rotation commands are represented by toolbar buttons. Unfortunately, the icons on these buttons don't work well. They're very similar to each other, and the arrow doesn't stand out (poor **contrast**). The icon tells a little story, showing before and after representations of a simplified abstract object. That's not such a bad thing in general, but it obscures the important differences between the two icons and forces you to study them carefully to figure out what they mean. Worse, the **mapping** is backwards: the Rotate Right button (with the right-pointing arrow) actually appears on the left.

The Snapfish web site (for storing and printing digital photo albums) has a neat solution to this problem. It does away with the notion of rotating entirely; instead, you just click on the side of the photo that you want to be on top. A little head-and-shoulders icon provides an **affordance** for clicking, while reminding about the heads-up orientation. This interface is neat because the controls are **directly mapped** to their effect (the side of the image that becomes the top). There's no need to mention right or left, clockwise or counterclockwise, or 90 or 180 degrees. The rotation is done by direct manipulation of the image itself. The labels are unfortunate – particularly the unreadable upside-down label! -- but new idioms often need extra help at first.

Nielsen's Heuristics

- **Meet expectations**
  1. Match the real world
  2. Consistency & standards
  3. Help & documentation
- **User is boss**
  4. User control & freedom
  5. Visibility of system status
  6. Flexibility & efficiency
- **Errors**
  7. Error prevention
  8. Recognition, not recall
  9. Error reporting, diagnosis, and recovery
- **Keep it simple**
  10. Aesthetic & minimalist design

Recall these 10 heuristics we discussed in an earlier lecture.

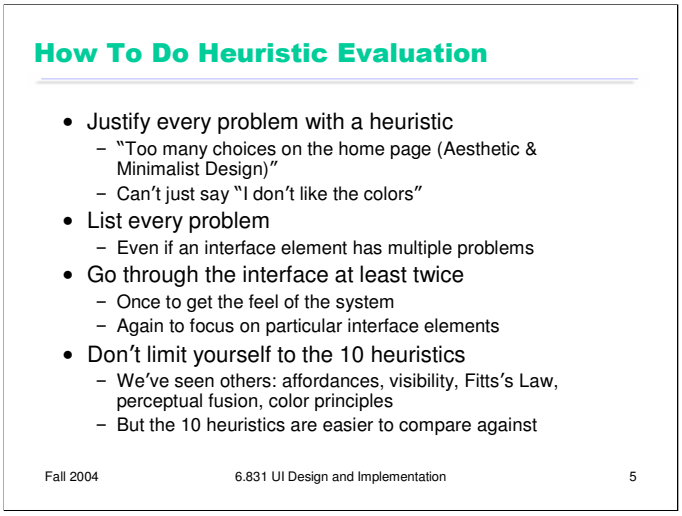


One application of these 10 heuristics is a usability inspection process called **heuristic evaluation**. Heuristic evaluation was originally invented by Jakob Nielsen, and you can learn more about it on his web site. Nielsen has done a number of studies to evaluate the effectiveness of heuristic evaluation. Those studies have shown that heuristic evaluation's cost-benefit ratio is quite favorable; the cost per problem of finding usability problems in an interface is generally cheaper than alternative methods.

Heuristic evaluation is an inspection method. It is performed by a usability expert – someone who knows and understands the heuristics we've just discussed, and has used and thought about lots of interfaces.

The basic steps are simple: the evaluator inspects the user interface thoroughly, judges the interface on the basis of the heuristics we've just discussed, and makes a list of the usability problems found – the ways in which individual elements of the interface deviate from the usability heuristics.

The Hall of Fame and Hall of Shame discussions we have at the beginning of each class are informal heuristic evaluations. In particular, if you look back at previous lecture notes, you'll see that most of the usability problems are justified by appealing to a heuristic.



Let's look at heuristic evaluation from the evaluator's perspective. That's the role you'll be adopting in the next homework, when you'll serve as heuristic evaluators for each others' computer prototypes.

Here are some tips for doing a good heuristic evaluation. First, your evaluation should be grounded in known usability guidelines. You should justify each problem you list by appealing to a heuristic, and explaining how the heuristic is violated. This practice helps remove most of the (inevitable) subjectivity involved in inspections: You can't just say "that's an ugly yellow color." (If it's really yucky, you *should* pass that subjective opinion back to the design team, but you'll be forced to identify it as subjective if you can't find a heuristic to justify it.)

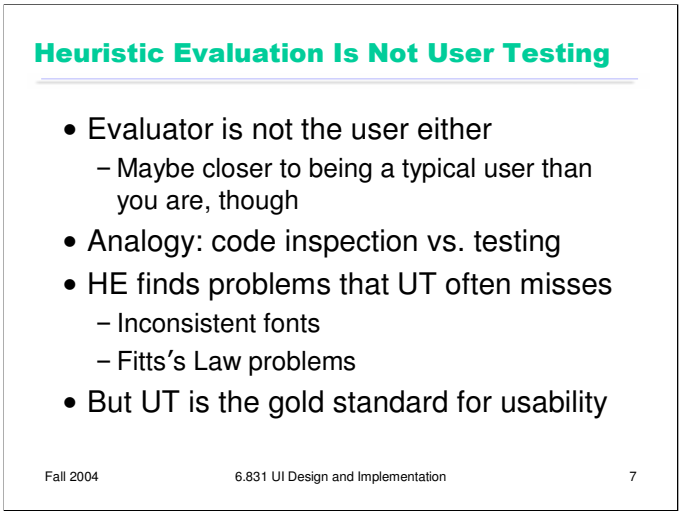
List every problem you find. If a button has several problems with it – inconsistent placement, bad color combination, confusing label – then each of those problems should be listed separately. Some of the problems may be more severe than others, and some may be easier to fix than others. It's best to get all the problems on the table in order to make these tradeoffs.

Inspect the interface at least twice. The first time you'll get an overview and a feel for the system. The second time, you should focus carefully on individual elements of the interface, one at a time.

Finally, although you have to justify every problem with a guideline, you don't have to limit yourself to the Nielsen 10. We've seen a number of specific usability principles that can serve equally well: affordances, visibility, Fitts's Law, perceptual fusion, color guidelines, graphic design rules are a few. The Nielsen 10 are helpful in that they're a short list that covers a wide spectrum of usability problems. For each element of the interface, you can quickly look down the Nielsen list to guide your thinking.

Let's try it on an example. Here's a partial heuristic evaluation of the screen shown above. Can you find any other usability issues?

1. Shopping cart icon is not balanced with its background whitespace (Aesthetic & minimalist design)
2. **Good:** user is greeted by name (Visibility of system status)
3. Red is used both for help messages and for error messages (Consistency, Match real world)
4. "There is a problem with your order", but no explanation or suggestions for resolution (Error reporting)
5. ExtPrice and UnitPrice are strange labels (Match real world)
6. Remove Hardware button inconsistent with Remove checkbox (Consistency)
7. "Click here" is unnecessary (Aesthetic & minimalist design)
8. No "Continue shopping" button (User control & freedom)
9. Recalculate is very close to Clear Cart (Error prevention)
10. "Check Out" button doesn't look like other buttons (Consistency, both internal & external)
11. Uses "Cart Title" and "Cart Name" for the same concept (Consistency)
12. Must recall and type in cart title to load (Recognition not recall, Error prevention, Flexibility & efficiency)

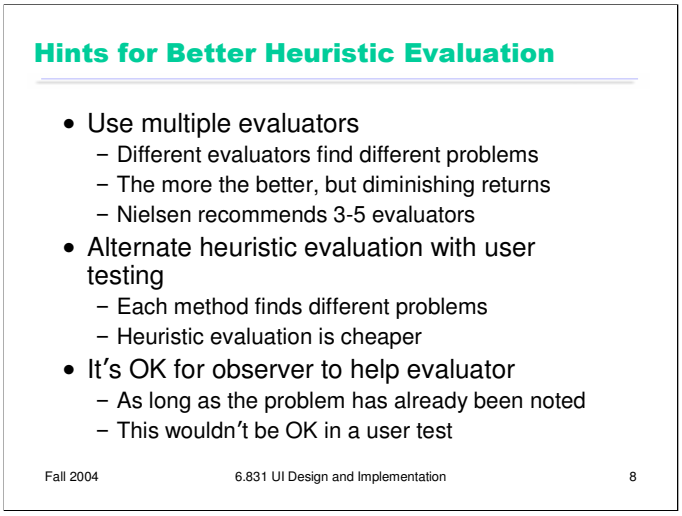


Heuristic evaluation is only one way to evaluate a user interface. User testing -- watching users interact with the interface – is another. User testing is really the gold standard for usability evaluation. An interface has usability problems only if real users have real problems with it, and the only sure way to know is to watch and see.

A key reason why heuristic evaluation is different is that an evaluator is not a typical user either! They may be closer to a typical user, however, in the sense that they don't know the system model to the same degree that its designers do. And a good heuristic evaluator tries to think like a typical user. But an evaluator knows too much about user interfaces, and too much about usability, to respond like a typical user.

So heuristic evaluation is not the same as user testing. A useful analogy from software engineering is the difference between code inspection and testing.

Heuristic evaluation may find problems that user testing would miss (unless the user testing was extremely expensive and comprehensive). For example, heuristic evaluators can easily detect problems like inconsistent font styles, e.g. a sans-serif font in one part of the interface, and a serif font in another. Adapting to the inconsistency slows down users slightly, but only extensive user testing would reveal it. Similarly, a heuristic evaluation might notice that buttons along the edge of the screen are not taking proper advantage of the Fitts's Law benefits of the screen boundaries, but this problem might be hard to detect in user testing.

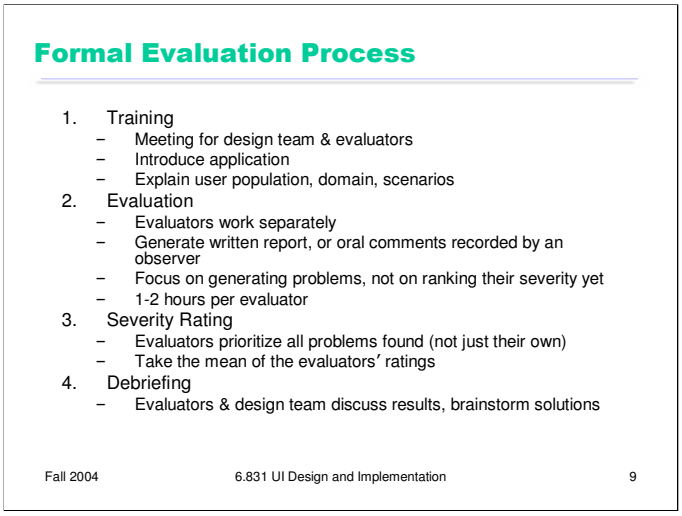


Now let's look at heuristic evaluation from the designer's perspective. Assuming I've decided to use this technique to evaluate my interface, how do I get the most mileage out of it?

First, use more than one evaluator. Studies of heuristic evaluation have shown that no single evaluator can find all the usability problems, and some of the hardest usability problems are found by evaluators who find few problems overall (Nielsen, "Finding usability problems through heuristic evaluation", CHI '92). The more evaluators the better, but with diminishing returns: each additional evaluator finds fewer new problems. The sweet spot for cost-benefit, recommended by Nielsen based on his studies, is 3-5 evaluators.

One way to get the most out of heuristic evaluation is to alternate it with user testing in subsequent trips around the iterative design cycle. Each method finds different problems in an interface, and heuristic evaluation is almost always cheaper than user testing. Heuristic evaluation is particularly useful in the tight inner loops of the iterative design cycle, when prototypes are raw and low-fidelity, and cheap, fast iteration is a must.

In heuristic evaluation, it's OK to help the evaluator when they get stuck in a confusing interface. As long as the usability problems that led to the confusion have already been noted, an observer can help the evaluator get unstuck and proceed with evaluating the rest of the interface, saving valuable time. In user testing, this kind of personal help is totally inappropriate, because you want to see how a user would really behave if confronted with the interface in the real world, without the designer of the system present to guide them. In a user test, when the user gets stuck and can't figure out how to complete a task, you usually have to abandon the task and move on to another one.



Here's a formal process for performing heuristic evaluation.

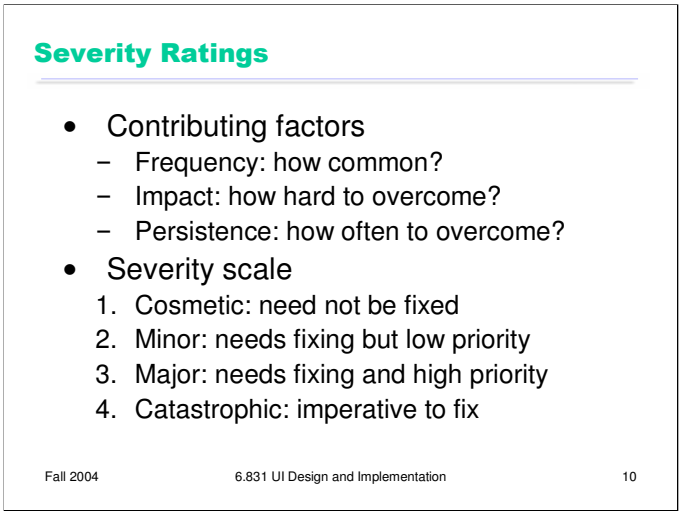
The training meeting brings together the design team with all the evaluators, and brings the evaluators up to speed on what they need to know about the application, its domain, its target users, and scenarios of use.

The evaluators then go off and evaluate the interface separately. They may work alone, writing down their own observations, or they may be observed by a member of the design team, who records their observations (and helps them through difficult parts of the interface, as we discussed earlier). In this stage, the evaluators focus just on generating problems, not on how important they are or how to solve them.

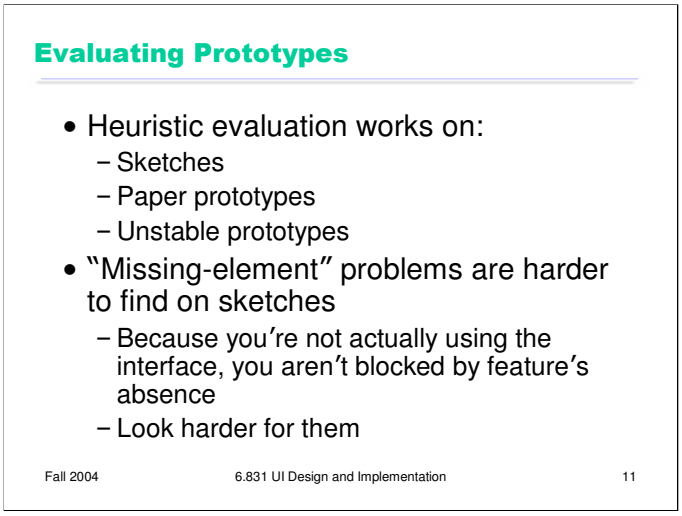
Next, all the problems found by all the evaluators are compiled into a single list, and the evaluators rate the severity of each problem. We'll see one possible severity scale in the next slide. Evaluators can assign severity ratings either independently or in a meeting together. Since studies have found that severity ratings from independent evaluators tend to have a large variance, it's best to collect severity ratings from several evaluators and take the mean to get a better estimate.

Finally, the design team and the evaluators meet again to discuss the results. This meeting offers a forum for brainstorming possible solutions, focusing on the most severe (highest priority) usability problems.

When you do heuristic evaluations in this class, I suggest you follow this ordering as well: first focus on generating as many usability problems as you can, then rank their severity, and then think about solutions.



Here's one scale you can use to judge the severity of usability problems found by heuristic evaluation. It helps to think about the factors that contribute to the severity of a problem: its **frequency** of occurrence (common or rare); its **impact** on users (easy or hard to overcome), and its **persistence** (does it need to be overcome once or repeatedly). A problem that scores highly on several contributing factors should be rated more severe than another problem that isn't so common, hard to overcome, or persistent.



A final advantage of heuristic evaluation that's worth noting: heuristic evaluation can be applied to interfaces in varying states of readiness, including unstable prototypes, paper prototypes, and even just sketches. When you're evaluating an incomplete interface, however, you should be aware of one pitfall. When you're just inspecting a sketch, you're less likely to notice missing elements, like buttons or features essential to proceeding in a task. If you were actually *interacting* with an active prototype, essential missing pieces rear up as obstacles that prevent you from proceeding. With sketches, nothing prevents you from going on: you just turn the page. So you have to look harder for missing elements when you're heuristically evaluating static sketches or screenshots.

Writing Good Heuristic Evaluations

- Heuristic evaluations must communicate well to developers and managers
- Include positive comments as well as criticisms
  - "Good: Toolbar icons are simple, with good contrast and few colors (minimalist design)"
- Be tactful
  - Not: "the menu organization is a complete mess"
  - Better: "menus are not organized by function"
- Be specific
  - Not: "text is unreadable"
  - Better: "text is too small, and has poor contrast (black text on dark green background)"

Here are some tips on writing good heuristic evaluations. First, remember your audience: you're trying to communicate to developers. Don't expect them to be experts on usability, and keep in mind that they have some ego investment in the user interface. Don't be unnecessarily harsh.

Although the primary purpose of heuristic evaluation is to identify problems, positive comments can be valuable too. If some part of the design is *good* for usability reasons, you want to make sure that aspect doesn't disappear in future iterations.

Suggested Report Format

- What to include:
  - Problem
  - Heuristic
  - Description
  - Severity
  - Recommendation (if any)
  - Screenshot (if helpful)

12. Severe: **User may close window without saving data** (error prevention)

    If the user has made changes without saving, and then closes the window using the Close button, rather than File >> Exit, no confirmation dialog appears.

    Recommendation: show a confirmation dialog or save automatically