Task-Centered User Interface Design

A Practical Introduction

by **Clayton Lewis** and **John Rieman**

Chapter 4: Evaluating the Design Without Users

- Example: Selecting Background Printing with the Mac Chooser
- 4.1 Cognitive Walkthroughs
  - Example: A Quick Cognitive Walkthrough
  - 4.1.1 Who should do a walkthrough, and when?
  - 4.1.2 What's needed before you can do a walkthrough?
  - HyperTopic: Common Mistakes in Doing a Walkthrough
  - 4.1.3 What should you look for during the walkthrough?
  - 4.1.4 What do you do with the results of the walkthrough?
  - Example: Cognitive Walkthrough of Setting Mac Background Printing
  - Credits and Pointers: Cognitive Walkthrough
- 4.2 Action Analysis
  - Table: Average times for computer interface actions
  - Example: Formal action analysis
  - 4.2.2 Back-of-the-Envelope Action Analysis
  - Example: Back-of-the-envelope action analysis
- 4.3 Heuristic Analysis
  - Table: Nielsen and Molich's Nine Heuristics
  - Example: One Evaluator's Heuristic Analysis of the Mac Background Printing Controls
  - Credits and Pointers: Heuristic Analysis
- 4.4 Chapter Summary and Discussion

Throughout this book we've emphasized the importance of bringing users into the interface design process. However, as a designer you'll also need to evaluate the evolving design when no users are present. Users' time is almost never a free or unlimited resource. Most users have their own work to do, and they're able to devote only limited time to your project. When users do take time to look at your design, it should be as free as possible of problems. This is a courtesy to the users, who shouldn't have to waste time on trivial bugs that you could have caught earlier. It also helps build the users' respect for you as a professional, making it more likely that they will give the design effort serious attention.

A second reason for evaluating a design without users is that a good evaluation can catch problems that an evaluation with only a few users may not reveal. The numbers tell the story here: An interface designed for a popular personal computer might be used by thousands of people, but it may be tested with only a few dozen users before beta release. Every user will have a slightly different set of problems, and the testing won't uncover problems that the few users tested don't have. It also won't uncover problems that users might have after they get more experience. An evaluation without users won't uncover all the problems either. But doing both kinds of evaluation significantly improves the chances of success.

In this chapter we describe three approaches to evaluating an interface in the absence of users. The first approach is the cognitive walkthrough, a task-oriented technique that fits especially well in the context of task-centered design. The second approach is action analysis, which allows a designer to predict the time that an expert user would need to perform a task, and which forces the designer to take a detailed look at

the interface. The third approach is heuristic evaluation, a kind of check-list approach that catches a wide variety of problems but requires several evaluators who have some knowledge of usability problems.

We'll describe these techniques and show how each one applies to the analysis of a single interface. The interface we'll look at is the "Chooser" in an early version of the Apple Macintosh operating system. The Chooser lets the user select printers and printer options. For our task-oriented evaluations, the task will be to turn on background printing.

The Chooser is an interesting example because it's part of a system that was designed with usability and simplicity as paramount goals. Nonetheless, we'll see that it has some potential problems. Some of those problems have been corrected in later versions of the Mac operating system; others haven't, possibly because the structure of the interface is too deeply embedded in the functionality of the system, or possibly because the changes would be too disruptive to existing users. (See the end of Appendix M for more thoughts on upgrading systems after they are in the field.)

Take a few minutes before you read the rest of this chapter to look over the following description of the Chooser and write down any problems you find.

Example: Selecting Background Printing with the Mac Chooser

We've presented this example in the rough format that a system designer might use to describe a suggested system to his colleagues. This is the point in an interface design when you should be using the techniques described in this chapter, either alone or as a group -- don't wait until after the system is implemented!

START

User is working with word processor. Task is to turn on background printing. Screen shows the current application window and the following menubar of pulldown menus:

```
-----------------------------------------
 @ File Edit Search Font Utilities
-----------------------------------------
```

("@" stands for the apple icon.)

ACTION 1: Pull down the apple menu.

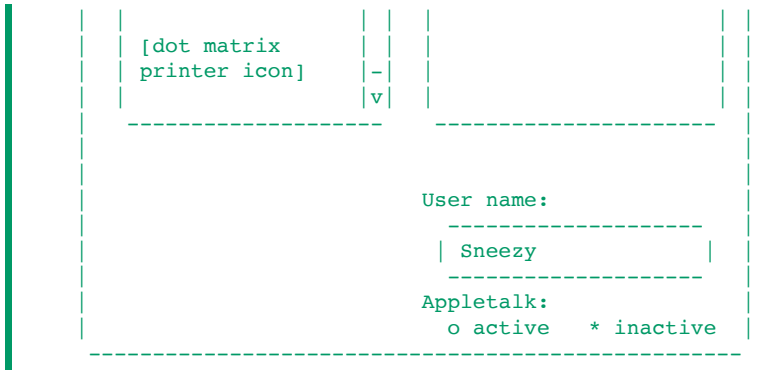
Apple pulldown menu looks like this:

```
| @ |
 ------------------
| About UltraWord  |
| ---------------- |
| Alarm Clock      |
| Calculator       |
| Chooser          |
| Control Panel    |
| Find File        |
| Keycaps          |
| Scrapbook        |
 ------------------
```

ACTION 2: Select "Chooser" from the menu.

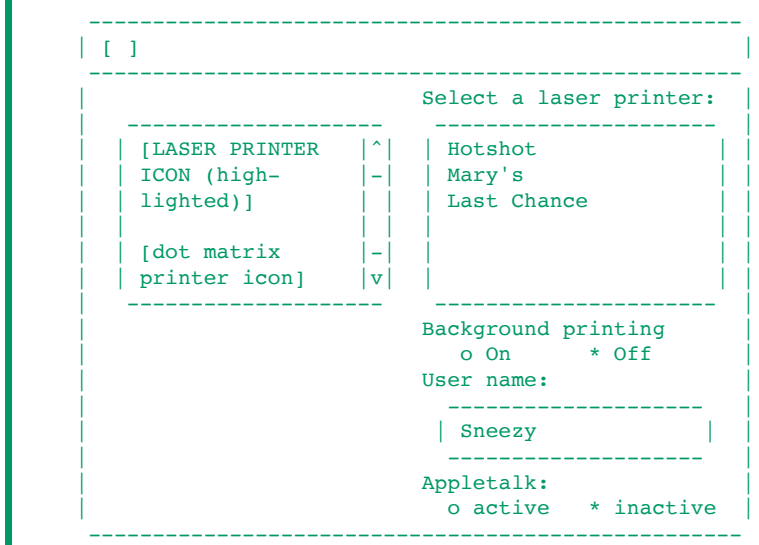
A dialog box appears:

```
 ------------------------------------------------
| [ ]                                            |
 ------------------------------------------------
|                                                |
|   --------------------    ----------------------  |
|   | [laser printer   |^|   |                    | |
|   | icon]            |-|   |                    | |
```



ACTION 3: Click on current printer type, which is Laser.

The laser printer icon highlights and new things appear in the dialog box:



ACTION 4: Click the On button under background printing.

The On button highlights and the Off button unhighlights.

ACTION 5: Click the Close Box in the upper left window.

The screen appears as it did at startup.

4.1 Cognitive Walkthroughs

The cognitive walkthrough is a formalized way of imagining people's thoughts and actions when they use an interface for the first time. Briefly, a walkthrough goes like this: You have a prototype or a detailed design description of the interface, and you know who the users will be. You select one of the tasks that the design is intended to support. Then you try to tell a believable story about each action a user has to take to do the task. To make the story believable you have to motivate each of the user's actions, relying on the user's general knowledge and on the prompts and feedback provided by the interface. If you can't tell a believable story about an action, then you've located a problem with the interface.

Example: A Quick Cognitive Walkthrough

Here's a brief example of a cognitive walkthrough, just to get the feel of the process. We're evaluating the interface to a personal desktop photocopier. A design sketch of the machine shows a numeric keypad, a "Copy" button, and a push button on the back to turn on the power. The machine automatically turns itself off after 5 minutes inactivity. The

task is to copy a single page, and the user could be any office worker. The actions the user needs to perform are to turn on the power, put the original on the machine, and press the "Copy" button.

In the walkthrough, we try to tell a believable story about the user's motivation and interaction with the machine at each action. A first cut at the story for action number one goes something like this: The user wants to make a copy and knows that the machine has to be turned on. So she pushes the power button. Then she goes on to the next action.

But this story isn't very believable. We can agree that the user's general knowledge of office machines will make her think the machine needs to be turned on, just as she will know it should be plugged in. But why shouldn't she assume that the machine is already on? The interface description didn't specify a "power on" indicator. And the user's background knowledge is likely to suggest that the machine is normally on, like it is in most offices. Even if the user figures out that the machine is off, can she find the power switch? It's on the back, and if the machine is on the user's desk, she can't see it without getting up. The switch doesn't have any label, and it's not the kind of switch that usually turns on office equipment (a rocker switch is more common). The conclusion of this single-action story leaves something to be desired as well. Once the button is pushed, how does the user know the machine is on? Does a fan start up that she can hear? If nothing happens, she may decide this isn't the power switch and look for one somewhere else.

That's how the walkthrough goes. When problems with the first action are identified, we pretend that everything has been fixed and we go on to evaluate the next action (putting the document on the machine).

You can see from the brief example that the walkthrough can uncover several kinds of problems. It can question assumptions about what the users will be thinking ("why would a user think the machine needs to be switched on?"). It can identify controls that are obvious to the design engineer but may be hidden from the user's point of view ("the user wants to turn the machine on, but can she find the switch?"). It can suggest difficulties with labels and prompts ("the user wants to turn the machine on, but which is the power switch and which way is on?"). And it can note inadequate feedback, which may make the users balk and retrace their steps even after they've done the right thing ("how does the user know it's turned on?").

The walkthrough can also uncover shortcomings in the current specification, that is, not in the interface but in the way it is described. Perhaps the copier design really was "intended" to have a power-on indicator, but it just didn't get written into the specs. The walkthrough will ensure that the specs are more complete. On occasion the walkthrough will also send the designer back to the users to discuss the task. Is it reasonable to expect the users to turn on the copier before they make a copy? Perhaps it should be on by default, or turn itself on when the "Copy" button is pressed.

Walkthroughs focus most clearly on problems that users will have when they first use an interface, without training. For some systems, such as "walk-up-and-use" banking machines, this is obviously critical. But the same considerations are also important for sophisticated computer programs that users might work with for several years. Users often learn these complex programs incrementally, starting with little or no training, and learning new features as they need them. If each task-oriented group of features can pass muster under the cognitive walkthrough, then the user will be able to progress smoothly from novice behavior to productive expertise.

One other point from the example: Notice that we used some features of the task that were implicitly pulled from a detailed, situated understanding of the task: the user is sitting at a desk, so she can't see the power switch. It would be impossible to include all relevant details like this in a written specification of the task. The most successful walkthroughs will be done by designers who have been working closely with real users, so they can create a mental picture of those users in their actual environments.

Now here's some details on performing walkthroughs and interpreting their results.

4.1.1 Who should do a walkthrough, and when?

If you're designing a small piece of the interface on your own, you can do your own, informal, "in your head" walkthroughs to monitor the design as you work. Periodically, as larger parts of the interface begin to coalesce, it's useful to get together with a group of people, including other designers and users, and do a walkthrough for a complete task. One thing to keep in mind is that the walkthrough is really a tool for developing the interface, not for validating it. You should go into a walkthrough expecting to find things that can be improved. Because of this, we recommend that group walkthroughs be done by people who are roughly at the same level in the company hierarchy. The presence of high-level managers can turn the evaluation into a show, where the political questions associated with criticizing someone else's work overshadow the need to improve the design.

4.1.2 What's needed before you can do a walkthrough?

You need information about four things. (1) You need a description or a prototype of the interface. It doesn't have to be complete, but it should be fairly detailed. Things like exactly what words are in a menu can make a big difference. (2) You need a task description. The task should usually be one of the representative tasks you're using for task-centered design, or some piece of that task. (3) You need a complete, written list of the actions needed to complete the task with the interface. (4) You need an idea of who the users will be and what kind of experience they'll bring to the job. This is an understanding you should have developed through your task and user analysis.

HyperTopic: Common Mistakes in Doing a Walkthrough

In teaching people to use the walkthrough method, we've found that there are two common misunderstandings. We'll point those out here, so you'll be on guard to avoid them.

First, many evaluators merge point (3) above into the walkthrough evaluation process. That is, they don't know how to perform the task themselves, so they stumble through the interface trying to discover the correct sequence of actions -- and then they evaluate the stumbling process.

There may be times when that approach is useful, but it misses an important part of the walkthrough method. In the walkthrough, you should START WITH A CORRECT LIST OF THE INDIVIDUAL ACTIONS needed to complete the given task: Click button "File", Type in "Smith Letter", Click button "OK," etc. If you have to explore the interface to identify those actions, fine -- but that's not the walkthrough. The walkthrough begins when you have the list of actions in hand. The reason for this approach is that, in the best of all worlds, the user should identify and perform the optimal action sequence. So the walkthrough looks at exactly that sequence. If the walkthrough shows that the user may have trouble identifying or performing one of the actions, your primary interest is not what the user will do when that problem arises -- what you really want to know is the fact that there is a problem here, which needs to be corrected.

The second point that many first-time users of the walkthrough method miss is to that the walkthrough DOES NOT TEST REAL USERS ON THE SYSTEM. Of course, watching real users try out the interface can be a valuable approach, and we discuss it in detail in Chapter 5. But the walkthrough is an evaluation tool that helps you and your co-workers apply your design expertise to the evaluation of the interface. Because you can imagine the behavior of entire classes of users, the walkthrough will often identify many more problems than you would find with a single, unique user in a single test session.

4.1.3 What should you look for during the walkthrough?

You've defined the task, the users, the interface, and the correct action sequence. You've gathered a group of designers and other interested folk together. Now it's time to actually DO THE WALKTHROUGH.

In doing the walkthrough, you try to tell a story about why the user would select each action in the list of

correct actions. And you critique the story to make sure it's believable. We recommend keeping four questions in mind as you critique the story:

1. Will users be trying to produce whatever effect the action has?
2. Will users see the control (button, menu, switch, etc.) for the action?
3. Once users find the control, will they recognize that it produces the effect they want?
4. After the action is taken, will users understand the feedback they get, so they can go on to the next action with confidence?

Here are a few examples -- "failure stories" -- of interfaces that illustrate how the four questions apply.

The first question deals with what the user is thinking. Users often aren't thinking what designers expect them to think. For example, one portable computer we've used has a slow-speed mode for its processor, to save battery power. Assume the task is to do some compute-intensive spreadsheet work on this machine, and the first action is to toggle the processor to high-speed mode. Will users be trying to do this? Answer: Very possibly not! Users don't expect computers to have slow and fast modes, so unless the machine actually prompts them to set the option, many users may leave the speed set at its default value -- or at whatever value it happened to get stuck in at the computer store.

The second question concerns the users' ability to locate the control -- not to identify it as the right control, but simply to notice that it exists! Is this often a problem? You bet. Attractive physical packages commonly hide "ugly" controls. One of our favorite examples is an office copier that has many of its buttons hidden under a small door, which has to be pressed down so it will pop up and expose the controls. If the task is, for example, to make double-sided copies, then there's no doubt that users with some copier experience will look for the control that selects that function. The copier in question, in fact, has a clearly visible "help" sheet that tells users which button to push. But the buttons are hidden so well that many users have to ask someone who knows the copier where to find them. Other interfaces that take a hit on this question are graphic interfaces that require the user to hold some combination of keys while clicking or dragging with a mouse, and menu systems that force the users to go through several levels to find an option. Many users will never discover what they're after in these systems without some kind of training or help.

The third question involves identifying the control. Even if the users want to do the right thing and the control is plainly visible, will they realize that this is the control they're after? An early version of a popular word processor had a table function. To insert a new table the user selected a menu item named "Create Table." This was a pretty good control name. But to change the format of the table, the user had to select a menu item called "Format Cells." The designers had made an analogy to spreadsheets, but users weren't thinking of spreadsheets -- they were thinking of tables. They often passed right over the correct menu item, expecting to find something called "Format Table." The problem was exacerbated by the existence of another menu item, "Edit Table," which was used to change the table's size.

Notice that the first three questions interact. Users might not want to do the right thing initially, but an obvious and well labeled control could let them know what needs to be done. A word processor, for example, might need to have a spelling dictionary loaded before the spelling checker could run. Most users probably wouldn't think of doing this. But if a user decided to check spelling and started looking through the menus, a "load spelling dictionary" menu item could lead them to take the right action. Better yet, the "check spelling" menu item could bring up a dialog box that asked for the name of the spelling dictionary to load.

The final question asks about the feedback after the action is performed. Generally, even the simplest actions require some kind of feedback, just to show that the system "noticed" the action: a light appears

when a copier button is pushed, an icon highlights when clicked in a graphical interface, etc. But at a deeper level, what the user really needs is evidence that whatever they are trying to do (that "goal" that we identified in the first question) has been done, or at least that progress has been made. Here's an example of an interface where that fails. A popular file compression program lets users pack one or more files into a much smaller file on a personal computer. The program presents a dialog box listing the files in the current directory. The user clicks on each file that should be packed into the smaller file, then, after each file, clicks the "Add" button. But there's no change visible after a file has been added. It stays in the list, and it isn't highlighted or grayed. As a result, the user isn't sure that all the files have been added, so he or she may click on some files again -- which causes them to be packed into the smaller file twice, taking up twice the space!

4.1.4 What do you do with the results of the walkthrough?

Fix the interface! Many of the fixes will be obvious: make the controls more obvious, use labels that users will recognize (not always as easy as it sounds), provide better feedback. Probably the hardest problem to correct is one where the user doesn't have any reason to think that an action needs to be performed. A really nice solution to this problem is to eliminate the action -- let the system take care of it. If that can't be done, then it may be possible to re-order the task so users will start doing something they know needs doing, and then get prompted for the action in question. The change to the "spelling dictionary" interaction that we described is one example. For the portable computer speed problem, the system might monitor processor load and ask if the user wanted to change to low speed whenever the average load was low over a 20 minute period, with a similar prompt for high speed.

Example: Cognitive Walkthrough of Setting Mac Background Printing

The task is to turn on background printing. The interface and action sequence are described in the box at the beginning of the chapter. We'll consider users who have some experience with this computer, but who have never set the background printing option.

The walkthrough analysis proceeds action by action.

The first action is to pull down the apple menu. (We might also aggregate the first two actions, saying "select Chooser from the apple menu." The analysis should reveal pretty much the same things.) We predict that users want to find a menu item that lets them select background printing -- that is, they want to do the right thing. The Apple menu is clearly visible, although it might not be clear that it's a menu to a first-time user. And since our users have used the Mac before, it's reasonable that they might look for a "system" function here. However they might also think background printing is a "print" option, and go looking under the "Print..." menu item under "File." This seems like it could be a problem. The low-level feedback is OK -- a menu drops. But it would be better if the menu had some items in it that clearly had to do with printing.

The second action is to select "Chooser" from the menu. The user is still trying to find the "printer" options. Will they realize that "Chooser" is the right menu item? It seems like they could just as reasonably select "Control Panel." Once again, we have to fault the labeling. Once the action is performed, however, the printer icons in the dialog box at least suggest that this is the right tactic. But notice: still nothing visible about "background printing."

Third action is click on the current printer type, which is laser printer. Why would the user want to do this? We can't come up with any good rationalization. Some users may not even see that the action is available, since icons in a scrolling window aren't a common Mac interface technique. But the feedback, assuming the user does click the icon, is finally a success: here's the background printing toggle we were looking for (assuming the user sees it below the list of printer names).

Fourth action is to click "On" for background printing. This seems to be problem free. It's exactly what the user wants to do;, the control is clearly visible; and the feedback -- the button becomes filled in -- is informative.

Final action is to close the Chooser window by clicking the close box in the upper left corner. We've been calling it a "dialog box," but it's actually a window filled with controls. This seems odd. Users with Mac experience will probably

think that they need to click the "OK" button, but there isn't one. And the close box is in the diagonally opposite corner from where the "OK" button should be, so it may be especially hard to find. This also raises a feedback question. Users expect dialog boxes to put options into effect when "OK" is clicked. Since there's no "OK" button, users may wonder if just closing the window activated the options.

Summary

There's a real start-up problem with this task. The user wants to set "background printing" but he or she has to go through three levels of controls that don't seem very closely related to that goal: the apple menu, the "chooser" menu item, and the printer type. It seems like all of these things need to be revised to make the path more obvious. A better solution might be to put the background printing toggle into a place where the user might already be looking: make it part of the dialog box that comes up when "Print" is selected from a "File" menu. The other noticeable problem with the interface is the use of a window instead of a dialog box with an "OK" button.

Credits and Pointers: Cognitive Walkthrough

The cognitive walkthrough was developed by Clayton Lewis, Peter Polson, Cathleen Wharton, and John Rieman. A description of the theoretical foundations of the method can be found in:

- Polson, P.G., Lewis, C., Rieman, J., and Wharton, C. "Cognitive walkthroughs: A method for theory-based evaluation of user interfaces." International Journal of Man-Machine Studies, 36 (1992), pp. 741-773.

The method has been tested in several situations. A summary and discussion of those tests, along with additional examples of interface success and failure stories, is given in:

- Wharton, C., Rieman, J., Lewis, C., and Polson, P. "The cognitive walkthrough: A practitioner's guide." In J. Nielsen and R.L. Mack (Eds.), Usability Inspection Methods, New York: John Wiley & Sons (1994).

4.2 Action Analysis

Action analysis is an evaluation procedure that forces you to take a close look at the sequence of actions a user has to perform to complete a task with an interface. In this book we'll distinguish between two flavors of action analysis. The first, "formal" action analysis, is often called "keystroke-level analysis" in HCI work. The formal approach is characterized by the extreme detail of the evaluation. The detail is so fine that, in many cases, the analysis can predict the time to complete tasks within a 20 percent margin of error, even before the interface is prototyped. It may also predict how long it will take a user to learn the interface. Unfortunately, formal action analysis isn't easy to do.

The second flavor of action analysis is what we call the "back of the envelope" approach. This kind of evaluation won't provide detailed predictions of task time and interface learnability, but it can reveal large-scale problems that might otherwise get lost in the forest of details that a designer is faced with. As its name implies, the back-of- the-envelope approach doesn't take a lot of effort.

Action analysis, whether formal or back-of-the-envelope, has two fundamental phases. The first phase is to decide what physical and mental steps a user will perform to complete one or more tasks with the interface. The second phase is to analyze those steps, looking for problems. Problems that the analysis might reveal are that it takes too many steps to perform a simple task, or it takes too long to perform the task, or there is too much to learn about the interface. The analysis might also reveal "holes" in the interface description -- things that the system should be able to do but can't. And it could be useful in

writing or checking documentation, which should describe the facts and procedures that the analysis has shown the user needs to know. 4.2.1 Formal Action Analysis

The formal approach to action analysis has been used to make accurate predictions of the time it takes a skilled user to complete tasks. To predict task times, the times to perform each small step of the task, physical or mental, are estimated, and those times are totalled. Most steps take only a fraction of a second. A typical step is a keystroke, which is why the formal approach is often called "keystroke- level analysis."

The predictions of times for each small step are found by testing hundreds of individual users, thousands of individual actions, and then calculating average values. These values have been determined for most of the common actions that users perform with computer interfaces. We summarize those values in the tables below. If an interface control isn't in the table, it might be possible to extrapolate a reasonable value from similar devices, or user testing might have to be done for the new control.

The procedure for developing the list of individual steps is very much like programming a computer. The basic task is divided into a few subtasks, like subroutines in a computer program. Then each of those subtasks is broken into smaller subtasks, and so on until the description reaches the level of the fraction-of-a-second operations listed in the table. The end result is a hierarchical description of a task and the action sequence needed to accomplish it.

Table: Average times for computer interface actions

[Based on detailed information in Judith Reitman Olson and Gary M. Olson, "The growth of cognitive modeling in human- computer interaction since GOMS," Human-Computer Interaction, 5 (1990), pp. 221-265. Many values given in this table are averaged and rounded.]

PHYSICAL MOVEMENTS		
Enter one keystroke on a standard keyboard:	.28 second	Ranges from .07 second for highly skilled typists doing transcription, to .2 second for an average 60-wpm typist, to over 1 second for a bad typist. Random sequences, formulas, and commands take longer than plain text.
Use mouse to point at object on screen	1.5 second	May be slightly lower -- but still at least 1 second -- for a small screen and a menu. Increases with larger screens, smaller objects.
Move hand to pointing device or function key	.3 second	Ranges from .21 second for cursor keys to .36 second for a mouse.
VISUAL PERCEPTION		
Respond to a brief light	.1 second	Varies with intensity, from .05 second for a bright light to .2 second for a dim one.
Recognize a 6-letter word	.34 second	
Move eyes to new location on screen (saccade)	.23 second	
MENTAL ACTIONS		

Retrieve a simple item from long-term memory	1.2 second	A typical item might be a command abbreviation ("dir"). Time is roughly halved if the same item needs to be retrieved again immediately.
Learn a single "step" in a procedure	25 seconds	May be less under some circumstances, but most research shows 10 to 15 seconds as a minimum. None of these figures include the time needed to get started in a training situation.
Execute a mental "step"	.075 second	Ranges from .05 to .1 second, depending on what kind of mental step is being performed.
Choose among methods	1.2 second	Ranges from .06 to at least 1.8 seconds, depending on complexity of factors influencing the decision.

Example: Formal action analysis

Here's an example of a formal action analysis, roughly in the style described by David Kieras (see Credits and Pointers).

The top-level goal is to turn on background printing. Our top-level analysis of the task is:

Method to accomplish goal of turning on background printing
- Step 1. Accomplish goal of making printer dialog box visible.
- Step 2. Accomplish goal of setting printer controls.
- Step 3. Accomplish goal of putting away printer dialog box.
- Step 4. Report goal accomplished

Here are some points to note about this first level. It doesn't include any actual, physical or mental actions. It is a "method" (the human version of a subroutine), and it's described in terms of goals that need to be accomplished. As we'll see, each of these goals will also be achieved with "methods." Also note that the three-step process is one example of a very common, generic Mac sequence: get a dialog box, set its controls, put it away. We could replace the word "printer" with "file" in Steps 1-3 and the procedure would work for saving a file. That says something good about the interface: it's consistent, so the user can learn one procedure and apply it to accomplish many things. Finally, note that there's an explicit step to report that the goal has been accomplished. This is parallel to the "return" call of a computer subroutine. In some procedures we might also make verifying the goal an explicit step.

Next we have to expand each step into methods. Step 1 expands into selecting from the menu. We'll skip the details of that and go on to expanding Step 2:

Method to accomplish goal of setting printer controls.
- Step 1. Accomplish goal of specifying printer type.
- Step 2. Accomplish goal of setting background printing control.
- Step 3. Report goal accomplished.

The methods we're describing are an explicit description of how we believe the user will conceptualize the task. Here we've had to make a decision, a judgement call, about what the user thinks. We've decided to break setting the printer controls into two steps, each with its own method. But another analyst might argue that the user will think of the task at this level as a single method or procedure, a single series of actions with the single goal of setting the background printing option in the dialog box. The results of the analysis could be somewhat different if that approach were taken.

We'll stay with the two-method approach and expand its first step. As it happens, there are two ways to specify the printer type. The user can click on the laser printer icon, as shown in the action sequence at the beginning of the chapter. Or, the user can type the first letter of the icon name, "Laser Printer." The formal analysis requires that we write a "selection rule," an if-then statement, that describes what conditions the user will consider to choose between the two techniques.

```
Selection rule for goal of specifying printer type.
   IF hand is on mouse
   THEN accomplish goal of specifying printer with mouse.
```

```
IF hands are on the keyboard
THEN accomplish goal of specifying printer with keyboard
Report goal accomplished
```

Now we'll expand the first of those options.

Method for accomplishing goal of specifying printer with mouse.

- Step 1. Recall current printer type.
- Step 2. Match type to name/icon in list.
- Step 3. Move to and click on icon.
- Step 4. Verify that icon is selected.
- Step 5. Report goal accomplished.

This is the lowest level of the hierarchy for this action. It describes low-level physical and mental actions at roughly the "keystroke" level. We can assign times to those actions as follows:

```
Recall current printer type:                  1.2 second
   - This is just recalling an item from long-term memory
Match type to name/icon list:                  1 second
   - This is a combination of moving the eyes, reading
     the name of the icon, and performing the mental step
     of matching.  It would take longer if there were
     more than two icons to choose from.
Move to and click on icon:                    1.5 second
   - Standard mouse movement time.  We know from the
     selection rule that the hand is already on the mouse.
System time to highlight icon:                 .1 second
   - We've just guessed at this.
Verify that icon is selected:                  .2 second
   - This is a combination of recognizing the change in
     intensity and making the mental step of interpreting
     it.
Report goal accomplished.                      .1 second
   -  A simple mental step.
```

This totals to 4 seconds. The selection rule that preceded the actions would add at least 1 second more. If you have a Mac handy and actually time yourself, you'll probably do the task a little faster. That's largely because you've already decided on a method, recalled your printer type, and have it's icon in mind (which will speed the matching process). But the method predicts that, on the average for a skilled user, setting the printer type will take about 5 seconds.

Points Raised by the Analysis

To complete the formal analysis we'd have to go down to the keystroke level for each of the steps in the process. We'd find that the full task would take on the order of 10 to 15 seconds on a fast Mac with a hard drive, but it could take 30 seconds or more on one of the early Macs with only a floppy drive. Most of the system time goes into opening and closing the dialog box. Time-wise, any small amount that might be shaved off the 8 to 10 seconds of user activity would be overshadowed by the savings that could be achieved by providing faster hardware.

But the analysis can highlight some problems in areas other than task time. We noted that evaluators might differ on whether "setting printer type" should be a separate goal or merged into the goal of setting background printing. If the evaluators have trouble making that decision, it's a good bet that users will have a related difficulty. The analysis doesn't really suggest a solution here, only a potential problem.

Another question that the analysis highlights is the need for a selection rule to decide between keyboard and mouse selection of the printer type. Is the rule we've proposed a good one? If the user has just used the mouse to choose from the menu, will the keyboard option ever be appropriate? This requires some further thought, since dialog boxes with selectable lists are a common feature in the interface. The selection technique should be consistent, but it should also be quick. Do the occasions when the keyboard will be used justify a 1 second decision penalty every time a list appears?

The analysis might also call into question the large number of steps needed to do this fairly simple task. Of course, as you can probably see, any analysis at this level produces a "large" number of steps, so we can't be sure there's a problem here without comparing the task to some other tasks of similar complexity. A related issue is the task's long-term memory requirements. The analysis makes it clear that the user must recall the printer type. Will the user know the printer type?

Summary

The formal analysis takes a lot of work, and it's not clear that it was worth it in this case, at least not for the timing results. But looking closely at the actions did reveal a few problems that go deeper than timing, including the way the user should think about setting printer type, the question of multiple selection methods for lists, and the amount of knowledge required to do this apparently trivial task.

A full-blown formal analysis of a complex interface is a daunting task. The example and the size of the time values in the table should give some idea of why this is so. Imagine you want to analyze two designs for a spreadsheet to see which is faster on a given task. The task is to enter some formulas and values, and you expect it to take the skilled user on the order of 10 minutes. To apply the formal action analysis approach you'll have to break the task down into individual actions, most of which take less than a second. That comes out to around 1000 individual actions, just to analyze a single 10-minute task! (There will probably be clusters of actions that get repeated; but the effort is still nontrivial.)

A further problem with formal analysis is that different analysts may come up with different results, depending on how they see the task hierarchy and what actions they predict a user will take in a given situation. (Will the user scan left, then down the spreadsheet? Down then left? Diagonally? The difference might be seconds, swamping other details.) Questions like this may require user testing to settle.

Because it's so difficult, we think that formal action analysis is useful only in special circumstances -- basically, when its high cost can be justified by a very large payoff. One instance where this was the case was the evaluation of a proposed workstation for telephone operators (see the article by Gray et al listed in Credits and Pointers, below). The phone company contracting the action analysis calculated that a savings of a few seconds in a procedure performed by thousands of operators over hundreds of thousands of calls would more than repay the months of effort that went into the evaluation.

Another place formal action analysis can be effective is for segments of the interface that users will access repeatedly as part of many tasks. Some examples of this are choosing from menus, selecting or moving objects in a graphics package, and moving from cell to cell within a spreadsheet. In each of these examples, a savings of a few tenths of a second in an interaction might add up to several minutes during an hour's work. This could justify a detailed analysis of competing designs.

In most cases, however, a few tenths of a second saved in performing an action sequence, and even a few minutes saved in learning it, are trivial compared to the other aspects of the interface that we emphasize in this book. Does the interface (and the system) do what the user needs, fitting smoothly into the rest of the user's work? Can the user figure out how the interface works? Does the interface's combination of controls, prompts, warning messages, and other feedback allow the user to maintain a comfortable "flow" through a task? If the user makes an error, does the system allow graceful recovery? All of these factors are central, not only to productivity but also to the user's perception of the system's quality. A serious failure on any of these points isn't going to be countered by shaving a few seconds off the edges of the interface.

4.2.2 Back-of-the-Envelope Action Analysis

The back-of-the-envelope approach to action analysis foregoes detailed predictions in an effort to get a quick look at the big picture. We think this technique can be very valuable, and it's easy to do. Like the formal analysis, the back-of- the-envelope version has two phases: list the actions, then think about them. The difference is that you don't need to spend a lot of time developing a detailed hierarchical breakdown of the task. You just list a fairly "natural" series of actions and work with them.

A process that works well for listing the actions is to imagine you are explaining the process to a typical user. That means you aren't going to say, "take your hand off the keyboard and move it to the mouse," or "scan down the menu to the 'chooser' item." You'll probably just say, "select 'chooser' from the apple menu." You should also put in brief descriptions of mental actions, such as "remember your password," or "convert the time on your watch to 24-hour time."

Once you have the actions listed there are several questions you can ask about the interface:

- Can a simple task be done with a simple action sequence?
- Can frequent tasks be done quickly?
- How many facts and steps does the user have to learn?
- Is everything in the documentation?

You can get useful answers to all these questions without going into fraction-of-a-second details. At the action level you'd use in talking to a user, EVERY ACTION TAKES AT LEAST TWO OR THREE SECONDS. Selecting something from a menu with the mouse, entering a file name, deciding whether to save under a new name or the old one, remembering your directory name -- watch over a user's shoulder sometime, or videotape a few users doing random tasks, and you'll see that combined physical and mental time for any one of these actions is a couple of seconds on a good day, three or four or even ten before morning coffee. And you'll have to start measuring in minutes whenever there's any kind of an error or mistake.

By staying at this level of analysis, you're more likely to keep the task itself in mind, along with the user's work environment, instead of getting lost in a detailed comparison of techniques that essentially do the same thing. For example, you can easily use the back-of-the-envelope results to compare your system's proposed performance with the user's ability to do the same task with typewriters, calculators, and file cabinets.

This kind of action analysis is especially useful in deciding whether or not to add features to an interface, or even to a system. Interfaces have a tendency to accumulate features and controls like a magnet accumulates paperclips in a desk drawer. Something that starts out as a simple, task-oriented action sequence can very quickly become a veritable symphony of menus, dialog boxes, and keystrokes to navigate through the options that various people thought the system should offer. Often these options are intended as "time savers," but the user ends up spending an inordinate amount of time just deciding which time saver to use and which to ignore. (One message you should take away from the table of action times is that it takes real time to decide between two ways of doing something.)

A few quick calculations can give you ammunition for convincing other members of a development team which features should or should not be added. Of course, marketing arguments to the contrary may prevail: it often seems that features sell a program, whether or not they're productive. But it's also true that popular programs sometimes become so complicated that newer, simpler programs move in and take over the low end of the market. The newcomers may even eventually displace the high-functionality leaders. (An example of this on a grand scale is the effect of personal computers on the mainframe market.)

Example: Back-of-the-envelope action analysis

Here's an example of an interface where a quick, informal action analysis shows real problems, which were gradually alleviated as the interface evolved. (We'll spare you from reading a second action analysis of the Mac chooser.) The interface is the controls to a 35-mm camera. When these cameras first became popular in the 1950s, the action sequence needed to take a picture went something like this:

```
- take light meter out of pocket
- make sure film speed is set correctly on light meter
```

```
      -- remember: what speed of film is in the camera
  - aim light meter at scene
  - read light value from needle on light meter
  - adjust calculator dial on light meter to light value
  - the calculator shows several possible combinations of
    f-stop and shutter speed, all of which give the
    correct exposure.  So...
     -- remember: big f-stop number means small lens opening
     -- remember: small lens opening means more depth of field
     -- remember: big shutter speed number means short exposure
     -- remember: short exposure means moving things aren't blurred
     -- decide: which combination of f-stop and speed to use
  - set the f-stop on the camera lens
  - set the shutter speed on the camera speed dial
  - remove the lens cap
  - cock the shutter
  - aim the camera
  - focus the camera
     -- remember: if lens aperture is small, depth of field is shallow
  - press the shutter release
  - advance the film
```

(Whew!)

Starting with the action list shown above, the camera designers could make rough estimates, or observe one or two users with a mock-up, and find that most actions in the list would take at least a couple of seconds. Some might take several seconds. These ballpark results would predict that it could take on the order of 30 seconds to take a photograph, and that's for a skilled user!

The analysis should make it clear that the interface would be hard to operate for several reasons:

- It takes too long to take a picture.
- There are too many actions to be learned.
- Some of the actions have to be "understood" -- for example, what IS depth of field? This makes the interface harder to learn and slower to use.
- Every action is an opportunity for error, and almost any error can ruin the final result.

The analysis would also show that more than half the time was spent metering the light and setting the exposure, a very system-oriented activity that many camera users don't want to think about.

We don't know if there was ever a back-of-the-envelope action analysis done on 35-mm cameras, but the development of the camera interface can be broadly described as an ongoing effort to combine or eliminate the steps described above, with the exposure-setting functions being the first to go. The ultimate realization of this effort is the modern "point- and-shoot" pocket 35-mm. The camera reads the film speed off the film cassette, light-meters the scene, selects a combination of shutter speed and f-stop, and turns on a flash if needed. (We didn't even mention the half-dozen or more additional actions needed to do flash photography before electronics!) Focusing with the point-and-shoot camera is automatic, and a motor cocks the shutter and winds the film. All that's left for the user is to aim and press the shutter release.

However, another look at the example illustrates some of the shortcomings of any action analysis, formal or informal. The modern, fully automated 35-mm camera isn't the only kind of camera people buy today. Many people buy cameras that are not fully automated or that let the user override the automation. Some of those purchasers are experienced photographers who believe they can get better or more varied results manually. Some are people who just like to have a lot of controls on their cameras. Whatever the reason, these contextual and market factors won't show up in an action analysis. But they can be discovered by designers who spend time getting to know the users.

Credits and Pointers: Action analysis

The quantitative, engineering-style analysis of actions was initially developed by Stuart Card, Thomas Moran, and Alan Newell, based on theories of cognitive psychology that Newell produced with Herbert Simon. The Card, Moran, and Newell approach is referred to as GOMS modelling (for Goals, Operators, Methods, and Selection). It's described in detail in:

- Card, S., Moran, T., and Newell, A. "The Psychology of Human-Computer Interaction." Hillsdale, New Jersey: Lawrence Erlbaum, 1983.

Important extensions to the work, including verifications and additions to the list of action times, have been made by a number of researchers. An excellent overview of this work is given in:

- Olson, Judith Reitman, and Olson, Gary M. "The growth of cognitive modeling in human-computer interaction since GOMS." Human-Computer Interaction, 5 (1990), pp. 221-265.

The detailed action analysis on telephone operator workstations is described in:

- Gray, W.D., John, B.E., Stuart, R., Lawrence, D., Atwood, M.E. "GOMS meets the phone company: Analytic modeling applied to real-world problems." Proc. IFIP Interact'90: Human-Computer Interaction. 1990, pp. 29-34.

The back-of-the-envelope analysis we describe draws on observations by David Kieras on the procedures for doing formal action analysis and interpreting its results. Kieras has worked to make the GOMS method easier to learn and more reliable. If you need to learn more about formal action analysis, a good place to start would be:

- Kieras, D.E. "Towards a practical GOMS model methodology for user interface design." In M. Helander (Ed.), "Handbook of Human-Computer Interaction" Amsterdam: Elsevier Science (North-Holland), 1988.

4.3 Heuristic Analysis

Heuristics, also called guidelines, are general principles or rules of thumb that can guide design decisions. As soon as it became obvious that bad interfaces were a problem, people started proposing heuristics for interface design, ranging from short lists of very general platitudes ("be informative") to a list of over a thousand very detailed guidelines dealing with specific items such as menus, command names, and error messages. None of these efforts has been strikingly successful in improving the design process, although they're usually effective for critiquing favorite bad examples of someone else's design. When the short lists are used during the design process, however, a lot of problems get missed; and the long lists are usually too unwieldy to apply. In addition, all heuristics require that an analyst have a fair amount of user interface knowledge to translate the general principles into the specifics of the current situation.

Recently, Jacob Nielsen and Rolf Molich have made a real breakthrough in the use of heuristics. Nielsen and Molich have developed a short list of general heuristics, and more importantly, they've developed and tested a procedure for using them to evaluate a design. We give the details of that procedure below, but first we want to say something about why heuristics, which are not necessarily a task-oriented evaluation technique, can be an important part of task- centered design.

The other two evaluation methods described in this chapter, the cognitive walkthrough and action analysis, are task- oriented. That is, they evaluate an interface as applied to a specific task that a user would be doing with the interface. User testing, discussed in chapter 6, is also task oriented. Task-oriented evaluations have some real advantages. They focus on interface problems that occur during work the user would actually be doing, and they give some idea of the importance of the problems in the context of the job. Many of the problems they reveal would only be visible as part of the sequence of actions needed to complete the task. But task-oriented evaluations also have some shortcomings. The first shortcoming is coverage: There's almost never time to evaluate every task a user would perform, so some action sequences and often some controls aren't evaluated. The second shortcoming is in identifying cross-task

interactions. Each task is evaluated standing alone, so task-oriented evaluations won't reveal problems such as command names or dialog-box layouts that are done one way in one task, another way in another.

Task-free evaluation methods are important for catching problems that task-oriented methods miss. Both approaches should be used as the interface develops. Now, here's how the heuristic analysis approach works.

Nielsen and Molich used their own experience to identify nine general heuristics (see table, below), which, as they noted, are implicit or explicit in almost all the lists of guidelines that have been suggested for HCI. Then they developed a procedure for applying their heuristics. The procedure is based on the observation that no single evaluator will find every problem with an interface, and different evaluators will often find different problems. So the procedure for heuristic analysis is this: Have several evaluators use the nine heuristics to identify problems with the interface, analyzing either a prototype or a paper description of the design. Each evaluator should do the analysis alone. Then combine the problems identified by the individual evaluators into a single list. Combining the individual results might be done by a single usability expert, but it's often useful to do this as a group activity.

Table: Nielsen and Molich's Nine Heuristics

- **Simple and natural dialog** - Simple means no irrelevant or rarely used information. Natural means an order that matches the task.
- **Speak the user's language** - Use words and concepts from the user's world. Don't use system-specific engineering terms.
- **Minimize user memory load** - Don't make the user remember things from one action to the next. Leave information on the screen until it's not needed.
- **Be consistent** - Users should be able to learn an action sequence in one part of the system and apply it again to get similar results in other places.
- **Provide feedback** - Let users know what effect their actions have on the system.
- **Provide clearly marked exits** - If users get into part of the system that doesn't interest them, they should always be able to get out quickly without damaging anything.
- **Provide shortcuts** - Shortcuts can help experienced users avoid lengthy dialogs and informational messages that they don't need.
- **Good error messages** - Good error messages let the user know what the problem is and how to correct it.
- **Prevent errors** - Whenever you write an error message you should also ask, can this error be avoided?

The procedure works. Nielsen and Molich have shown that the combined list of interface problems includes many more problems than any single evaluator would identify, and with just a few evaluators it includes most of the major problems with the interface. Major problems, here, are problems that confuse users or cause them to make errors. The list will also include less critical problems that only slow or inconvenience the user.

How many evaluators are needed to make the analysis work? That depends on how knowledgeable the evaluators are. If the evaluators are experienced interface experts, then 3 to 5 evaluators can catch all of the "heuristically identifiable" major problems, and they can catch 75 percent of the total heuristically identifiable problems. (We'll explain what "heuristically identifiable" means in a moment.) These experts might be people who've worked in interface design and evaluation for several years, or who have several years of graduate training in the area. For evaluators who are also specialists in the specific domain of the interface (for example, graphic interfaces, or voice interfaces, or automated teller interfaces), the same results can probably be achieved with 2 to 3 evaluators. On the other hand, if the evaluators have no

interface training or expertise, it might take as many as 15 of them to find 75 percent of the problems; 5 of these novice evaluators might find only 50 percent of the problems.

We need to caution here that when we say "all" or "75 percent" or "50 percent," we're talking only about "heuristically identifiable" problems. That is, problems with the interface that actually violate one of the nine heuristics. What's gained by combining several evaluator's results is an increased assurance that if a problem can be identified with the heuristics, then it will be. But there may still be problems that the heuristics themselves miss. Those problems might show up with some other evaluation method, such as user testing or a more task-oriented analysis.

Also, all the numbers are averages of past results, not promises. Your results will vary with the interface and with the evaluators. But even with these caveats, the take-home message is still very positive: Individual heuristic evaluations of an interface, performed by 3 to 5 people with some expertise in interface design, will locate a significant number of the major problems.

To give you a better idea of how Nielsen and Molich's nine heuristics apply, one of the authors has done a heuristic evaluation of the Macintosh background printing controls (see Box).

Example: One Evaluator's Heuristic Analysis of the Mac Background Printing Controls

Remember, for an effective heuristic analysis, SEVERAL evaluators should check out the interface independently and combine their notes into a single problem list.

Since heuristic analysis is usually a task-free approach, the Mac background printing controls would probably be presented to the evaluator as part of an overall description of the Chooser, or perhaps of the Mac system interface as a whole. If just the Chooser were being checked, the design description might specify how the dialog box would be brought up through the Apple menu (shown in Steps 1 and 2) and it might have sketches of how the box would look before and after a printer was specified (shown in Steps 3 and 4). It would also give some description of how to use the controls in the dialog box.

Here's my evaluation. This piece of the interface is small enough that I can just go through the nine heuristics one at a time and look for possible problems anywhere in the design. (For a larger design, I might want to step through the nine heuristics several times, once for each manageable chunk of the interface.)

The first heuristic is "simple and natural dialog." OK, the idea of this design seems to be that I select what kind of printer I'm going to use, then specify some options. I guess that will work, but why do I have to select the kind of printer and say which one? Couldn't I tell it which one and let the system figure out whether it's a laser or dot-matrix printer?

Second heuristic: "speak the user's language." I can see an effort to do this. They've called it the "Chooser" instead of "Output options." But an even better name might be "Printer" -- parallel to "Alarm Clock" and "Calculator." I don't know what "Appletalk" is, though. Is there some more common name for this? And why don't the printers have names like "central office" or "library," indicating where they are?

Heuristic: "Minimize user memory load." This heuristic primarily applies to things the user has to notice in one step and remember one or more steps later. Everything stays pretty visible here, so the memory load is low. But I do have to remember what kind of printer I'm using and what its name is.

Heuristic: "Be consistent." I don't think I've seen a scrolling list of clickable icons in any other dialog box. Do the icons add anything here? Maybe there should just be a scrolling list of text entries.

Heuristic: "Provide feedback." Design seems good to me. Something happens whenever I do anything. Of course, there's no evidence of what I've done after the dialog box goes away. But that seems OK here. I don't mind having some options set "behind the scenes," as long as I can check them whenever I want.

Heuristic: "Provide clearly marked exits." Well, looking at the dialog box, I see that it has a "close box" in the upper left. I kind of expect an "OK" and a "Cancel" button in a dialog box. Maybe I should have noted this under consistency with the rest of the interface? Anyway, it might confuse some people.

Heuristic: "Provide shortcuts." I suppose the usual Mac menu-selection shortcuts work -- like typing "M" to select "Mary's" printer." Should there be other shortcuts? I dunno. Depends on whether people need to do something really often, like toggling AppleTalk or Background printing. Need some more task analysis here.

Heuristic: "Good error messages." Current design doesn't describe any error messages. I guess I'd like users to get an error if they selected a printer that wasn't connected to the computer. Need to check on that. Also, what if they turn off background printing while something is in the queue?

Heuristic: "Prevent errors." Same comments as for error messages, except it would be better if the user wasn't given options that wouldn't work. So if there isn't an ImageWriter connected, they shouldn't be able to select it -- it should probably be grayed out, or not there at all.

Summary

Based on what I've noticed in the heuristic analysis, I'd like to see some changes. I'd want the menu item to be called "Printer," and I'd want the dialog box to give me a list of currently available printer locations to choose from. The system should figure out what kind of printer it is. Also, I'd use an "OK" and a "Cancel" button instead of a window close box.

(Some discussion with the designers would probably raise the point that selecting a printer type effects how a document is formatted, even if a printer isn't connected. I'd have to argue that this really isn't "speaking the user's language." I think most users would expect to find formatting options under the format menu in the application.)

Credits and Pointers: Heuristic Analysis

The heuristic analysis technique was developed by Jacob Nielsen and Rolf Molich. Tests of the technique's effectiveness are described in:

- Nielsen, J. and Molich, R. "Heuristic evaluation of user interfaces." Proc. CHI'90 Conference on Human Factors in Computer Systems. New York: ACM, 1990, pp. 249-256.
- Nielsen, J. "Finding usability problems through heuristic evaluation." Proc. CHI'92 Conference on Human Factors in Computer Systems. New York: ACM, 1992, pp. 373-380.

More detailed descriptions of the nine heuristics are given in:

- Molich, R. and Nielsen, J. "Improving a human-computer dialogue: What designers know about traditional interface design." Communications of the ACM, 33 (March 1990), pp. 338-342.
- Nielsen, J. "Usability Engineering." San Diego, CA: Academic Press, 1992.

4.4 Chapter Summary and Discussion

We've looked at three methods for evaluating an interface without users. If you scan through the "Summary" sections at the ends of the examples, you'll see that each method uncovered different problems. The cognitive walkthrough identified problems with finding the controls and suggested that they be moved to the Print dialog box. A one-person heuristic analysis noted the same problem and some others, and suggested that the controls should be relabeled. The formal action analysis suggested a vague question about the printer-type selection and made it clear that there were a lot of actions needed to do this simple task. A back-of-the- envelope action analysis might have given similar results, without details of how long the task took.

Even the combined result of the three techniques didn't catch all the problems. That's partly because there are kinds of problems that none of these methods will catch, and partly because no individual analysis is perfect. One example of something none of the methods caught, although both the cognitive walkthrough

and the heuristic analysis noted related problems, is that "Chooser" and "Control Panel" have similar functions (setting options about the system) and also similar names (starting with C, two o's in the middle). It's likely that users will sometimes slip and pick the wrong one even when they know which one they're after, especially since items are alphabetized in the Apple menu. This problem could surface in user testing, or at least in beta testing. Another problem, which another heuristic analyst might have caught, is that the dialog box doesn't let users know which printer type is selected when it first opens. A user who isn't sure what kind of printer is being used may change to the wrong type, after which printing may not work at all.

Here's a recommendation on how to use the various methods in the design process. In general, the cognitive walkthrough will give you the best understanding of problems it uncovers, and we recommend thinking through the interface in walkthrough terms as you develop it. When a substantial part of the interface is complete, heuristic analysis is a good way to catch additional problems -- but you need several evaluators to do it well. Back-of-the-envelope action analysis makes a good "sanity check" as the interface becomes more complex, and it's also useful early in system design to help decide whether a system's features will pay back the effort of learning and using them. Formal action analysis is probably only appropriate for very special cases, as we described in that section. All the methods need to be combined with user testing, which we discuss in the next chapter.