CHAPTER 8

Usability analysis and inspection

Chapter objectives:

We are likely to be concerned about questions of usability at every stage in the design of an interactive system. Usability analysis tools enable us to answer these questions on paper, without building and testing a prototype. Here we look at:

- The kinds of usability analyses needed during design
- The two-stage approach to analysis in which a walkthrough stage precedes the analysis of performance
- Specific analysis techniques: GOMS, Cognitive Walkthrough and Heuristic Evaluation.

8.1 Introduction

We perform analyses when we are faced with the question, 'How well will this design do the job?' It is a question that arises in many forms of words during the course of designing interactive systems; for example:

> Will the operator be able to handle emergency telephone calls faster than before?
>
> Have we simplified the design of this ticket machine to a point where people will use it successfully on their first attempt?

Is the small size of this screen target going to result in a significant number of errors in selecting it?

If the user invokes this command by mistake, will he or she find the escape route?

Will the word-processor user remember that there are three different ways of changing the properties of a formatting style?

Is it so difficult to change the layouts of menus that hardly any users will bother?

Once the system is set up to support work-groups of a particular size and structure, how much effort is involved in changing the system to support changes in the group?

How many of the people who try the system will actually continue to use it?

These are questions about *usability* – the system's ability to support the user's activities. They arise constantly during design, right from the problem definition stage through requirements specification to detailed design. It is quite common to encounter usability questions even during the system's implementation and introduction to use. In this chapter and the next we will be looking at methods of answering these questions.

Before we start, we should note that questions about interactive system designs cover many other issues besides usability. There are other levels of technology involved, as Chapter 2 pointed out. Questions about the software's performance are especially likely to arise, as will questions about hardware cost and performance, maintenance, safety, and so forth. But these questions are not exclusive to interactive systems, and so we won't be concerned with them here. Good software-engineering textbooks, such as Sommerville's (1994), are available covering methods of analysis and evaluation in each of these areas.

8.2 Answering usability questions

Questions about usability are rarely straightforward to answer. None of the questions listed above, for example, would be at all easy to answer off-the-cuff. Usability is a system property that we can rarely determine by a quick calculation. On the contrary, it involves conducting an analysis or experiment. Depending on the complexity of the design and the desired accuracy of the answer, we may be in for an exercise lasting anywhere from ten minutes to six months. Before we enter into this exercise we need a clear idea of the form of answer we want and how we are going to derive it.

8.2.1 Two approaches to measuring usability

Broadly speaking, measurements of usability can be conducted in either of two ways:

- **Analytically**, by performing a simulation of how the user's activities will be performed;
- **Empirically**, by building a prototype and testing it in the hands of users.

These are two very different approaches to answering questions about usability. Analytical techniques can sometimes be carried out quite quickly and informally, using 'back-of-envelope' methods. They don't involve real users, and so there's much less organizing to do. Nevertheless they can on occasions involve a considerable amount of work – see, for example, the analyses reported in Card *et al.* (1983) and Diaper (1989a).

Empirical measurements, on the other hand, almost always involve a lot of prior planning, and rely on careful conduct and analysis of the experiment (Hix and Hartson, 1993). Users need to be 'signed up' as subjects, and testing must be arranged at times when the users are available. Thus even the simplest investigation can take several days to conduct, and will need to be planned with care. But the results usually justify the time and effort invested, because it is possible to address aspects of usability that lie beyond the scope of analytical methods.

This chapter is devoted solely to *analytical methods*, and to methods of *usability inspection* aimed at discovering flaws in the design that will affect the system's usability. The next two chapters cover empirical methods of usability evaluation.

8.3 Analysis as a two-stage process

The point was made earlier that usability analysis involves more than just a single quick calculation. The basic reason for this is that we are concerned with the performance of a human activity, and our analysis must therefore be applied to each step in the activity's performance. We cannot do a split-second usability analysis any more than we can instantly account for the money we spent during a week's vacation.

Rather, we are likely to perform the analysis in two stages. First we determine the sequence or method by which the activity is performed; then we analyse the steps in the sequence to determine usability measures. These two stages are shown in Figure 8.1.

As we can see, each stage of analysis operates on a *model* of the user's activity, and results in a refinement of the model. We start with a general model, perhaps just indicating the goal of the activity. From this, and from a description of the system's functionality, we build a model of a sequence of actions. From the activity sequence model we build a model that identifies usability properties, such as the time taken to perform each step, or the steps where errors may occur.

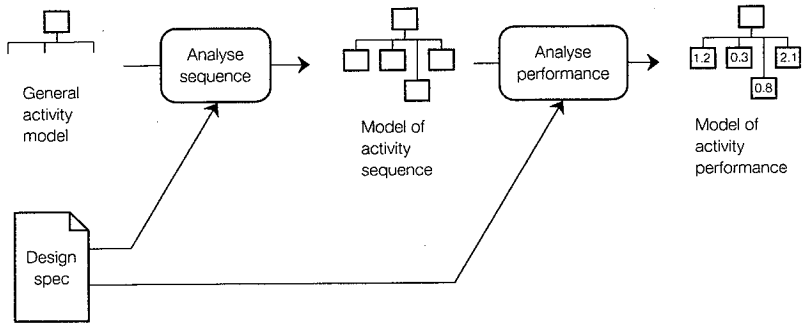

Figure 8.1 Stages of analysis, building successively more refined and specific models.

8.3.1 Determining the sequence by walkthrough

The first stage of analysis, in which we build a model of the activity sequence, is an important one. If we get the sequence wrong, this invalidates our usability analyses. But how do we determine the sequence?

Sequence analysis relies largely on **walkthrough methods**, by which each step in the sequence is determined from the state of the interaction after the previous step, and from the user's current goal. Thus if the user's goal is to correct the spelling of a word in a document, her first step is likely to be to find the word in question. With the word in view, her next step may well be to select the word. With the word selected, her next step is probably to retype it. We have just 'walked through' the sequence, building a model of how we expect the activity to be performed (Figure 8.2).

It's important to realize that, in many interactive contexts, there is no single walkthrough sequence – the activity may be performed in a wide variety of different ways. For example, the sequence by which someone edits a six-page technical report can vary considerably; the sequence by which someone else reads the report will be almost entirely unpredictable. Sometimes it is necessary to model a number of alternative sequences in order to check for possible usability problems. These may take the form of **benchmarks**, that is, activities that will be used to compare the performance of different design solutions.

8.3.2 Determining the sequence empirically

It may be easier and more reliable to determine the sequence of operation empirically, by studying users. This is preferable if, for example, the activity involves special skills and is performed by highly trained operators.

Suppose we were interested in how a patent officer might conduct

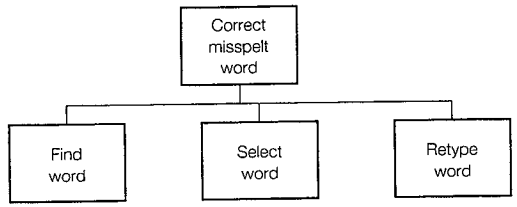

Figure 8.2 Task model of correcting a misspelt word, generated by walkthrough.

searches for previous patents that may relate to a current patent application. We wouldn't attempt to work this out on our own with the aid of walkthrough methods. After all, what do we know about searching through patent archives? So we would conduct a user study, observing patent officers at work and interviewing them about their methods. Then we would build a model of the sequence, or sequences, of activity performance.

8.3.3 Folding in the usability analysis

Once we know the sequence of the user's activity we can perform a more detailed usability analysis. This may take the form of a separate stage of analysis, as in the keystroke-level technique described in the next section.

It is also appropriate, and sometimes convenient, to fold in the usability analysis with the walkthrough analysis. It is often the case that usability is affected by the juxtaposition of steps, not by each step in isolation.

8.3.4 Methods of analysis

The first two methods of analysis discussed in this chapter are variants of the basic walkthrough method. In effect, they each adapt the basic method to the needs of particular kinds of design questions. The first method, *Keystroke-Level Analysis*, provides measures of speed of operation by trained users. The second, *Cognitive Walkthrough*, is tuned to assessing users' learning problems in systems that they walk up and use. By way of contrast, we conclude the chapter with a non-walkthrough method, *Heuristic Evaluation*, but we will see that even this method involves consideration of sequences of operation.

8.4 Analysis techniques based on the GOMS model

The theory of the human information processor, outlined in Chapter 3, has given rise to a number of tools and techniques known collectively as methods of GOMS *analysis*. One particular GOMS technique, keystroke-level

analysis, was introduced at the start of Chapter 3. GOMS techniques are particularly useful in cases where we know the sequence of operation and want to find out how quickly the sequence can be performed by an experienced operator. With the aid of GOMS we can walk through the sequence, assigning approximate times to each of the steps, and thus calculate an overall performance time. It sounds simple, and in many instances it is.

GOMS is not always appropriate for analysing designs. It does not give accurate answers when the method of operation isn't known or the user is inexperienced. Its most effective use by far is in predicting speed of performance, because here it can draw on a large body of empirical laws and results, some of which were summarized in Chapter 3. In addition GOMS can sometimes be used to predict errors or choice of method.

8.4.1 The GOMS model

The GOMS model of task performance has four parts. Each part relates to one of the letters in the acronym, which stand for 'Goals, Operators, Methods and Selection rules'. These are the four kinds of variable that affect the outcome of the analyses. Thus:

- The **Goal** defines the end-state that the user is trying to achieve; for example, the goal may be to change to boldface the two words 'The cat' of the sentence 'The cat sat on the mat'. Top-level goals are subdivided into subgoals, and each of these is then analysed separately. Thus we can divide this task into two subtasks: selecting the two words, and setting the selected text to boldface.
- **Operators** are the basic actions available to the user for performing a task, such as moving the mouse pointer, clicking the mouse button, pressing a key.
- **Methods** are sequences of operators, or procedures, for accomplishing a goal. A method for selecting the words 'The cat' is to move the pointer to 'The', hold the mouse button down, move to 'cat' and release the button. A method of setting the selected words to boldface is to hold down the CONTROL key and press 'B'. These two methods are both employed in the sequence shown in Figure 8.3.
- **Selection rules** are invoked when there is a choice of method. The user interface doesn't always offer a choice. However, there is an alternative to the boldfacing method shown in the bottom half of Figure 8.3, involving the use of a pull-down menu to select the **Bold** reformatting command, and this is shown in Figure 8.4.

GOMS models of task performance can support analysis in a number of ways. They are at their simplest and most effective when predicting speeds of task performance in those cases where the method of operation

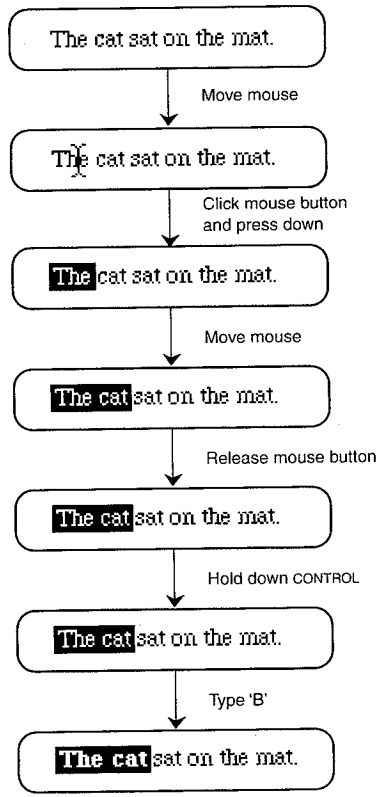

Figure 8.3 A method for selecting the words 'The cat' and changing them to boldface.

is known, that is, when selection rules are not an issue. This is done through *Keystroke-Level Analysis*, a technique described in the next section. Another kind of analysis concerns method selection: by estimating the speed of alternative methods we can compare them, and judge whether the user is likely to prefer one to another.

8.4.2 Keystroke-Level Analysis

The purpose of the Keystroke-Level Model is to predict the user's speed of execution of tasks. It can be used in situations where the user's method of performing the task is known: for example, it could be applied to

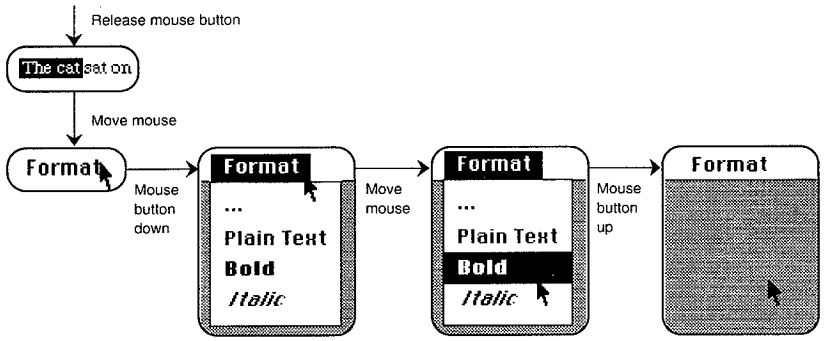

Figure 8.4 An alternative method, using a pull-down menu, for changing the selected words 'The cat' to boldface after selecting them. Courtesy Apple Computer.

dialling a telephone number, since this tends to follow a standard sequence very closely (lift handset, check for dial tone, press buttons in sequence, wait). It could not be applied to the task of drawing a freehand portrait with the aid of a painting program, since everyone does this differently.

The basis of the method is to divide each task-performance method into components, and assign execution times in seconds to each component, as shown in Table 8.1. These times have been derived from repeated experiments, but they can be shown to correspond to predictions derived from the information processing model and from extensions to it, such as Fitts' Law.

Use of the Keystroke-Level Model involves applying certain rules, or heuristics, in the introduction of **mental preparation** components. These **M**-components should usually, but not always, be introduced before keystrokes or pointing operations that are not part of a sequence. The rules proposed by Card *et al.* (1983) are given in Figure 8.5.

Keystroke assignments are simple and quick to apply to sequences of actions, but some practice is needed in order to apply the rules for placing **M** operators correctly. Two worked examples follow, and Case Study A describes a more detailed application of GOMS modelling. A number of worked examples can also be found in Card *et al.* (1983).

A simple keystroke-level comparison

We can use keystroke-level analysis to compare the two methods of selecting a pair of words and setting them to boldface, shown in Figures 8.3 and 8.4. The method of Figure 8.3 is analysed in the following table. Here the rules of Figure 8.5 have been applied to remove unneeded **M** operators

Table 8.1 Performance times for keystroke-level operators; from Card *et al* (1983).

Operator	*Description and remarks*	*Time (sec)*
K	PRESS KEY OR BUTTON. Pressing the SHIFT or CONTROL key counts as a separate K operation. Time varies with the typing skill of the user; the following shows the range of typical values:	
	Best typist (135 wpm)	.08
	Good typist (90 wpm)	.12
	Average skilled typist (55 wpm)	.20
	Average non-secretary typist (40 wpm)	.28
	Typing random letters	.50
	Typing complex codes	.75
	Worst typist (unfamiliar with keyboard)	1.20
P	POINT WITH MOUSE TO TARGET ON A DISPLAY. The time to point varies with distance and target size according to Fitt's Law, ranging from .8 to 1.5 sec, with 1.1 being an average. This operator does *not* include the (.2 sec) button press that often follows. Mouse pointing time is also a good estimate for other efficient analogue pointing devices, such as joysticks (see Chapter 7).	1.10
H	HOME HAND(S) ON KEYBOARD OR OTHER DEVICE.	.40
D(n_D,l_D)	DRAW n_D STRAIGHT-LINE SEGMENTS OF TOTAL LENGTH l_D CM. This is a very restricted operator; it assumes that drawing is done with the mouse on a system that constrains all lines to fall on a square .56cm grid. Users vary in their drawing skill; the time given is an average value.	$.9n_D + .16l_D$
M	MENTALLY PREPARE.	1.35
R(t)	RESPONSE BY SYSTEM. Different commands require different response times. The response time is counted only if it causes the user to wait.	t

(the number of the rule applied is shown in the right-hand column). A value of 0.50 seconds has been chosen for **K** operators, since these are mostly random keys:

Select words			
Reach for mouse	**H**	0.40	
Point to word 'The' with mouse	**P**	1.10	
Double-click and hold down mouse button	**K**	0.60	(1)
Move mouse to word 'cat'	**P**	1.10	
Finish selection by releasing mouse button	**K**	0.60	(1)

Begin with a method of encoding that includes all physical operations and response operations. Use Rule 0 to place candidate **M**'s, and then cycle through Rules 1 to 4 for each **M** to see where it should be deleted.

Rule 0. Insert **M**'s in front of all **K**'s that are not part of text or numeric argument strings proper (e.g., text or numbers). Place **M**'s in front of all **P**'s that select commands (not arguments).

Rule 1. If an operator following an **M** is *fully anticipated* in an operator just previous to **M**, then delete the **M** (e.g., **PMK** → **PK**).

Rule 2. If a string of **MK**'s *belongs to a cognitive unit* (e.g.,the name of a command), then delete all **M**'s but the first.

Rule 3. If a **K** is a *redundant terminator* (e.g., the terminator of a command immediately following the terminator of its argument), then delete the **M** in front of it.

Rule 4. If a **K** *terminates a constant string* (e.g., a command name), then delete the **M** in front of it; but if the **K** terminates a variable string (e.g., an argument string) then keep the **M** in front of it.

Figure 8.5 Rules for placing **M** operators. From Card *et al.* (1983).

Set to boldface:			
Press CONTROL	**K**	0.60	(2)
Type 'b'	**K**	0.60	(2)
Release CONTROL	**K**	0.60	(3)
Total		5.60 secs	

Note that no **H** operator is included before the CONTROL-b action, which can be given with the left hand only.

In comparison, the sequence of Figure 8.4, in which the **Format** pull-down menu is used, involves the following operators:

Select words (unchanged from previous example)			
Reach for mouse	**H**	0.40	
Point to word 'The' with mouse	**P**	1.10	
Double-click and hold down mouse button	**K**	0.60	(1)
Move mouse to word 'cat'	**P**	1.10	
Finish selection by releasing mouse button	**K**	0.60	(1)
Set to boldface:			
Point to Format menu with mouse	**P**	1.10	
Press and hold down mouse button	**K**	0.60	(1)
Move down to Bold	**P**	1.10	(1)
Release mouse button	**K**	0.60	(1)
Total		7.20 secs	

Thus the use of the keyboard 'short cut' does indeed reduce the task's performance time, by over a second and a half. This is due mainly to avoidance of two **P** operators.

Keystroke-Level Analysis: A second example

As a further example of a design problem where GOMS modelling is useful, consider the design of a method for generating hard-copy plots of data. This is a problem of choosing an efficient sequence of operation for a set of existing programs. The original plots are produced on the screen with a program called *plotgraph*, running in the UNIX environment, and the plots are printed by using the UNIX *xwd* and *xpr* utilities.

The first stage of analysis is to walk through the operating sequence, identifying each of the steps:

(1) Type the following to plot the data:

plotgraph <4-digit number> <3-digit number> <return>

The program takes an average of five seconds to produce the plot.

(2) Type the following to run *xwd*:

xwd –out abc <return>

The cursor changes within one second to a cross in readiness for selecting the window containing the plot.

(3) Select the window by pointing anywhere within it. The window image is dumped on the file *abc*, which takes six seconds.

(4) Type the following to print the image file:

xpr –device ps abc / lpr <return>

The *xpr* program responds within one second.

The Keystroke-Level predictions for task performance times are as follows. A time of 0.30 seconds is assigned to **K** operators. Again, numbers in the right-hand column refer to the rules in Figure 8.5.

(1) Run *plotgraph*:

Mentally prepare	**M**	1.35	
Type *plotgraph*	9**K**	2.70	
Type terminating SPACE	**K**	0.30	(4)
Type 4-digit number	4**K**	1.20	
Mentally prepare	**M**	1.35	
Type terminating SPACE	**K**	0.30	
Type 3-digit number	3**K**	0.90	
Mentally prepare	**M**	1.35	
Type terminating RETURN	**K**	0.30	
System response	**R**	5.00	
Subtask 1: *plotgraph*	Total	14.75 secs	

(2) Run *xwd*:

Mentally prepare	**M**	1.35	
Type *xwd* command	12**K**	3.60	
Type terminating RETURN	**K**	0.30	(4)
System response	**R**	1.00	
Subtask 2: *xwd*	Total	6.25 secs	

(3) Select window:

Reach for mouse	**H**	0.40	
Point to window with mouse	**P**	1.10	
Click mouse button	**K**	0.30	(1)
System response	**R**	6.00	
Subtask 3: window selection	Total	7.80 secs	

(4) Run *xpr*:

Mentally prepare	**M**	1.35	
Type *xpr* command	24**K**	7.20	
Type terminating RETURN	**K**	0.30	(4)
System response	**R**	1.00	
Subtask 4: *xpr*	Total	9.85 secs	

Adding together the subtotals, we arrive at the following total time for each printout:

Subtask 1: *plotgraph*	14.75 secs
Subtask 2: *xwd*	6.25 secs
Subtask 3: window selection	7.80 secs
Subtask 4: *xpr*	9.85 secs
Total	38.65 secs

The significance of this time prediction depends on the number of plots to be generated. Suppose we need 300 plots? This will take over three hours at the workstation. How can this be reduced?

A simple method of speeding up the task is to set up an alias, such as *p*, for the command *plotgraph*. This takes eight **K** operators, or 2.40 seconds, out of the first subtask, and reduces the overall time for 300 plots by 12 minutes. The search for other means of saving time is left as an exercise for the reader, but we can already see that the task must take at least 13 seconds each time, or over an hour for 300 plots, just in terms of the system's response time. It is the remaining two hours that we can hope to reduce.

8.5 Analysis by Cognitive Walkthrough

Cognitive Walkthroughs provide a method of analysing designs in terms of *exploratory learning*. They can be applied to designs for systems that will be used by people without any prior training, perhaps in a walk-up-and-use manner. They are also useful in the analysis of systems whose designs

have been changed or extended, because these changes and extensions may be encountered by users who have never been taught how to use them. In either case, the user must learn how to use the system by exploring its user interface. Cognitive Walkthroughs answer questions of the form, 'How successfully does this design guide the unfamiliar user through the performance of the task?'

We tend to use Cognitive Walkthroughs, then, when we want to assess operation by users who are exploring the system's user interface and learning as they go. Particular measures we will be looking for concern users' success rates in completing tasks, and their ability to recover from errors. We should not expect to measure users' speed of task performance. We need a fairly complete description of the user interface in order to conduct Cognitive Walkthroughs, because we need to cover all possible routes that the user may take. However, we don't need to know the user's sequence of operation, because the analysis itself helps us to discover what the sequence is likely to be.

8.5.1 The underlying model of exploratory learning

Analysis by Cognitive Walkthrough involves simulating the way users explore and gain familiarity with interactive systems, with the aid of the simple step-by-step model introduced in Chapter 3:

(0) The user starts with a rough plan of what he or she wants to achieve – a task to be performed;

(1) The user explores the system, via the user interface, looking for actions that might contribute to performing the task;

(2) The user selects the action whose description or appearance most closely matches what he or she is trying to do;

(3) The user then interprets the system's response and assesses whether progress has been made towards completing the task.

The analysis involves simulating steps 1, 2 and 3 at each stage of interaction, by asking questions of the form:

Q1: Will the correct action be made sufficiently evident to the user?

Q2: Will the user connect the correct action's description with what he or she is trying to do?

Q3: Will the user interpret the system's response to the chosen action correctly, that is, will the user know if he or she has made a right or a wrong choice?

The result of performing a Cognitive Walkthrough is usually to discover problems in these three areas, that is, where the questions receive a 'No' answer. Solutions to these problems are fed into the next iteration of design.

8.5.2 Cognitive Walkthrough: An example

To illustrate the use of Cognitive Walkthroughs, let us analyse the use of the rapid-transit ticket machine described in Chapter 6. The preliminary user interface design for the machine is reproduced in Figure 8.6.

We will work through an analysis of the machine's use by a first-time user. Let us suppose this user wishes to purchase a round-trip ticket to Dragon Plaza. We will add a complication: the traveller has only ten dollars in cash, but doesn't know this at the outset.

Our first step in the walkthrough is to answer the task-definition question:

Q0: What does the user want to achieve?

Answer: Purchase a round-trip ticket to Dragon Plaza.

Now we can enter into the walkthrough analysis itself. The initial display is as shown in Figure 8.6. We start by asking question Q1 about this display:

Q1: Will the correct action be made sufficiently evident to the user?

Answer: There are two possible correct actions, press the 'Dragon Plaza' button or press 'round-trip'. The design doesn't make this clear, for it instructs the user to choose the destination before indicating the journey type. This doesn't impede the user's progress, but it hides an available option. Thus we have identified **Design Flaw**

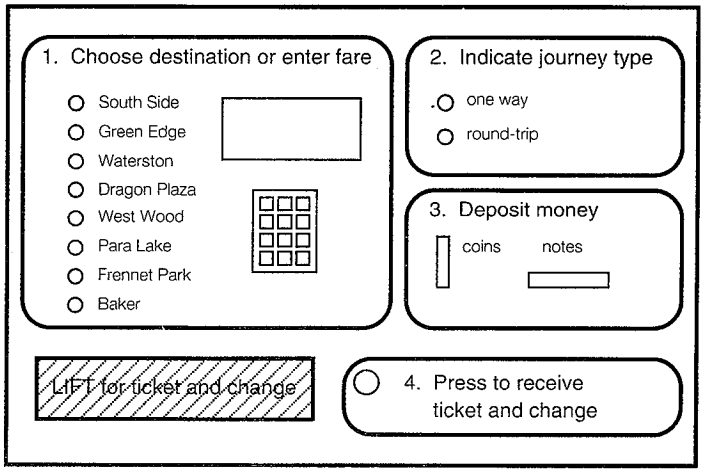

Figure 8.6 The design for a rapid-transit ticket machine, as developed in Chapter 6, to be analysed by Cognitive Walkthrough.

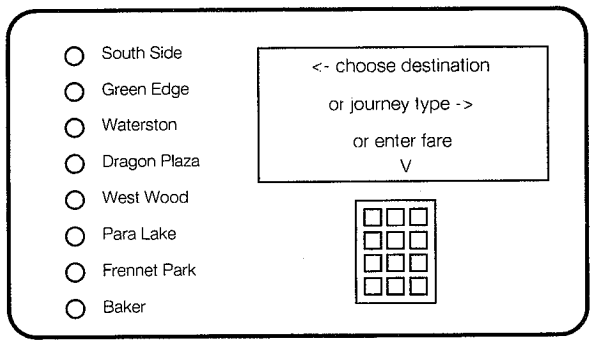

Figure 8.7 Making the range of methods more obvious to the user

no. 1: Option to indicate journey type first is not made sufficiently evident. One possible way to rectify this flaw might be to provide a prompt via a larger display (see Figure 8.7).

The user is thus likely to be aware of only one correct action, pressing the 'Dragon Plaza' button. Following the walkthrough analysis sequence, we will now ask questions Q2 and Q3 about this action:

Q2: Will the user connect the correct action's description with what he or she is trying to do?

Answer: Yes, the instructions for panel 1 and the button label will enable the user to make the connection.

Q3: Will the user interpret the system's response to the chosen action correctly, that is, will the user know if he or she has made a right or a wrong choice?

Answer: The machine will respond by lighting up the button pressed, as shown in Figure 8.8. This should appear to the user as confirmation of a correct action.

The user must now indicate the journey type, using panel 2. We apply the same walkthrough steps as before:

Q1: Will the correct action be made sufficiently evident to the user?

Answer: Yes. The correct action is to press the 'round-trip' button in panel 2, and the instructions labelling this panel make it clear that this is the next step.

Q2: Will the user connect the correct action's description with what he or she is trying to do?

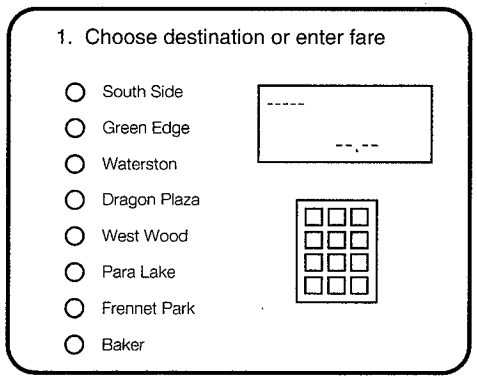

Figure 8.8 Confirming the user's choice of destination with a lighted button.

Answer: Yes, the instructions and the labels on the buttons make the connection very clear.

Q3: Will the user interpret the system's response to the chosen action correctly, that is, will the user know if he or she has made a right or a wrong choice?

Answer: Yes. The machine will respond by lighting up the 'round-trip' button and by displaying the journey type and fare, as shown in Figure 8.9. This provides confirmation of this and the previous action. If the user should press the 'one-way' button by mistake, this too will be made clear.

We'll now proceed to the next action, depositing money.

Q1: Will the correct action be made sufficiently evident to the user?

Answer: Yes. Again, the user's main source of assistance is the numbered sequence of instructions. According to this sequence the next step is to deposit money.

Q2: Will the user connect the correct action's description with what he or she is trying to do?

Answer: Yes, a request to deposit money is consistent with purchasing a ticket.

Q3: Will the user interpret the system's response to the chosen action correctly, that is, will the user know if he or she has made a right or a wrong choice?

Answer: Unclear. We need to know what kind of response the machine will provide to a correct action, that is, to depositing the first portion of the fare. If the machine merely swallows the money, the user will be little the wiser. According to Figure 8.9, there is no means of indicating receipt of the money, and thus no means for the user to keep track of the amount deposited. This could be considered **Design Flaw no. 2: No display of total money received**. A solution might be to extend the display as shown in Figure 8.10.

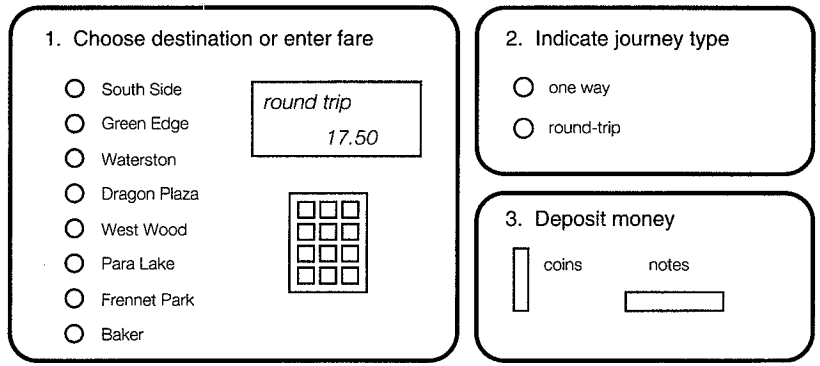

Figure 8.9 The machine's response to selecting the journey type.

The completion of the walkthrough of the normal ticket-purchase sequence is left as an exercise for the reader. Let us conclude this example by instead introducing the slight complication we've held in reserve – the

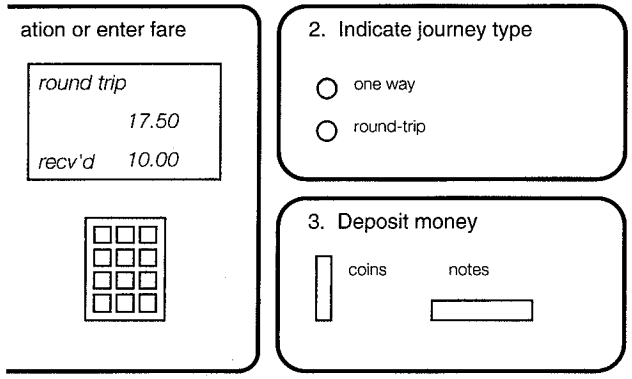

Figure 8.10 Improving the design to show the amount paid so far.

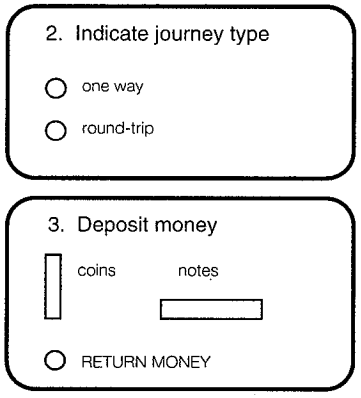

Figure 8.11 Adding the capability to retrieve money deposited.

passenger who has only ten dollars, and who discovers this only after depositing all of it.

Q1: Will the correct action be made sufficiently evident to the user?

Answer: No, because there is no action that the user can take to retrieve the money deposited. We have discovered **Design Flaw no. 3: No means of retrieving money deposited**. The design should include a 'return money' button as shown in Figure 8.11.

This initial analysis has been very useful, in pointing out three design flaws:

- The option to indicate the journey type before the destination is not made clear.
- No display is provided of the amount of money deposited.
- There is no means to retrieve money deposited.

The analysis will not stop there, however. A design such as this will need to be subjected to a full range of walkthrough analyses, covering all of the likely tasks that users will want to perform, under all likely conditions.

8.6 Heuristic Evaluation

The two methods discussed so far, GOMS analyses and Cognitive Walkthroughs, both have the disadvantage that they can be applied successfully only to certain kinds of design problems. We need methods of analysing designs in situations where the method of operation is not fully predictable, and where the user is not a complete novice. Indeed, a great many designs fall into this category. The method discussed in this section, *Heuristic Evaluation*, can be applied in these cases.

As its name suggests, Heuristic Evaluation is not a pure analysis technique. It might be thought of as 'analysis by a team of analysts using a variety of informal models'. A more useful descriptive term applied to Heuristic Evaluation is *Usability Inspection*: an inspection is carried out, and a list of problems that could affect usability is drawn up. Then, as with Cognitive Walkthroughs, the designer revises the solution to address the problems.

8.6.1 The method of evaluation

For its effect, the Heuristic Evaluation method relies on two techniques in combination.

First, it employs a **team of evaluators** rather than relying on one person to carry out the analysis. This has a number of advantages: in particular, a team of external evaluators can be more impartial than the designers themselves; and through a process of *aggregation* a compilation is made of the problems identified by each evaluator, with the result that a small team of perhaps four evaluators can produce a comprehensive list.

Second, a set of **design heuristics** is used to guide the evaluators. Heuristics can be thought of as general-purpose guidelines; the set recommended by Nielsen and Molich (1989) is shown in Figure 8.12. Any one of these heuristics is likely to apply only some of the time; for example, 'Be consistent' is sometimes inapplicable, as Grudin has pointed out (1989). The important role of the heuristics is to guide the analysis that the evaluators apply. 'Prevent errors', for example, would focus the evaluator on searching for errors, perhaps by scanning the design for features that the user might misinterpret. 'Provide short cuts' would suggest looking for frequently performed tasks that involve lengthy sequences of actions. Ultimately the evaluator will fall back on performing informal walkthroughs in response to many of the heuristics.

Simple and natural dialogue	Provide clearly marked exits
Speak the user's language	Provide short cuts
Minimize user memory load	Good error messages
Be consistent	Prevent errors
Provide feedback	

Figure 8.12 Usability heuristics used to guide a team of evaluators (from Nielson and Molich, 1989).

A number of advantages are claimed for the Heuristic Evaluation method (Nielsen, 1992). They include low cost, in comparison to other methods; intuitive to perform; no advance planning required, since the evaluations can be conducted by team members in isolation; and suitable for use early in the development process. Disadvantages are its focus on problems rather than solutions, and its tendency to encourage designers to strengthen the overall solution proposed rather than break away from it (Nielsen and Molich, 1990).

It should be added that Heuristic Evaluation is intrinsically **less repeatable** than analysis methods that are strongly theory-based. If the same design is offered to two evaluators, they may well deliver two quite different sets of design problems. If different descriptions of the same design are given to the same team, again, different lists of problems may be handed back. Through the use of a number of evaluators, however, and the aggregation of problems, the lack of repeatability is brought under control.

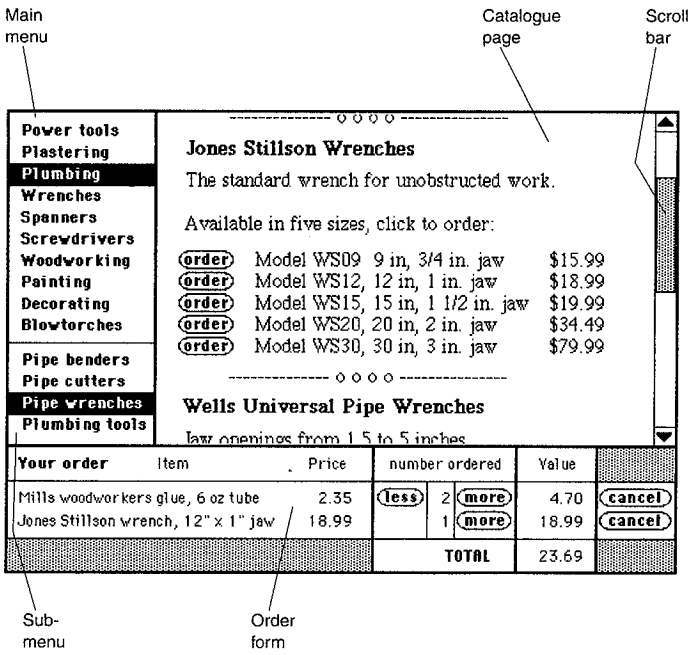

Figure 8.13 Online hardware catalogue user interface.

Example: Heuristic evaluation of an online hardware-store catalogue

The use of Heuristic Evaluation can be appreciated from a brief example of evaluating a design for an online catalogue for use by customers of a hardware store. Its purpose is to allow customers to place orders so that they can pay for them and collect them from a pick-up counter. A typical display layout is shown in Figure 8.13.

The catalogue is laid out as a two-level menu, the first level (shown in the top left of the screen) allowing the user to choose between broad categories of hardware, the second level (below the first) allowing access to individual types of product. To select a menu item, the user points to it with his or her finger.

Details of the products available are shown in the main window. Each item has an **order** target that can be selected by pointing. The scroll bar on the right can be used for moving up and down the list of available products.

When the user points to the **order** target, an entry is made in the bottom panel. The number ordered is set initially to 1, but may be altered with the **less** and **more** targets. As items are added, and the numbers ordered are changed, the total value of the purchase is shown at the bottom of the 'Value' column. The **cancel** target can be used to cancel the order for an item.

Here are some of comments that evaluators might make about this design, and the related heuristics:

Evaluator A: The menus aren't labelled, so it's hard for the user to know their purpose, or to understand the hierarchy (*Simple and natural dialogue*).

Evaluator C: I think the user will tend to confuse the menus, they look too much alike (*Simple and natural dialogue*).

Evaluator A: Why display the menus all the time anyway? While the user is searching the product descriptions, the menus aren't needed (*Simple and natural dialogue*).

Evaluator B: It would be good to have pictures of the products, that's what conventional catalogues have (*Speak the user's language*).

Evaluator D: The panel at the bottom is too small – people will order more than just two different items. They won't be able to remember all the items they've ordered (*Minimize user memory load*).

Evaluator B: The targets for ordering items are different from the menu targets, which are different again from the scroll-bar target. Shouldn't they be more alike? (*Be consistent*).

Evaluator C: How does the customer confirm the order? There's no button for doing this.

Evaluator A: What if the user accidentally cancels an order? They're going to have to go back to the catalogue to order it again (*Prevent errors*).

Evaluator B: It's more normal to show price and value alongside each other, and to place the number ordered on the left (*Speak the user's language*).

Some of these problems are relatively loosely connected with the heuristics. We should expect this, because there are always specialized considerations to take into account in any design problem. There are a number of special heuristics to consider in the design of menu-based dialogues, for example. We will return to the subject of heuristics and guidelines in Chapter 15.

8.7 Conclusion

A range of analysis methods are available to the designer of interactive systems. In this chapter we have looked at GOMS analysis techniques, Cognitive Walkthroughs and Heuristic Evaluation. There are other methods in addition to these, some of which are found in Nielsen (1993) and Preece (1994).

To be realistic, however, the choice of analysis method is often likely to be very restricted, because most of the available methods are themselves of restricted applicability. GOMS, for example, demands a definition of the methods of operation, while Cognitive Walkthrough applies primarily to analysis of learning by exploration. Although the designer will sometimes have the opportunity to use two or more methods of analysis and compare the results, the opposite extreme can also arise: no suitable method of analysis can be found. In these situations, evaluation must be postponed until a prototype can be built and tested. The next chapter discusses this use of prototype evaluation as an alternative to analysis.

Exercises

(1) What is the difference between analytical and empirical methods of assessing usability? Why are some assessments made analytically and some empirically?

(2) Why is it usually necessary to perform a walkthrough analysis before attempting to apply some performance analysis techniques? Why is this not needed in the case of Cognitive Walkthroughs?

(3) Conduct a Cognitive Walkthrough analysis of performing the following tasks with the aid of the ticket machine of Figure 8.6: (a) travelling as far as possible for $10; (b) buying a round-trip ticket for $14.55; (c) after selecting a round-trip ticket to West Wood and depositing $10, deciding to buy a one-way ticket instead.

(4) Online library catalogue systems often pose problems for first-time users because they must offer a wide range of functions. Conduct a Cognitive Walkthrough analysis of an online catalogue to which you have access.

(5) Calculate the time for scrolling a Macintosh or Microsoft Windows window. Ignore the value of **R**, the system response time, in calculating times for the following tasks: (a) moving to the end of the displayed document; (b) moving up one screen-full; (c) moving from near the end of the document to near the beginning, and then stepping down two screens-full.

(6) What difference will faster or slower typing make to the second example of Section 8.4.2?

(7) Suppose, in the example of Figure 8.4, the CONTROL-b formatting command is changed to CONTROL+SHIFT+b; in other words, the user must hold down CONTROL and SHIFT with the left hand, and type 'b' with the right. What effect will this have on the overall task performance time? Does this still represent a 'short cut'?

(8) In the second keystroke-level analysis example, how much time would be saved in generating 300 plots by (a) reducing the plot-file name to one character instead of three? (b) invoking *xwd* and *xpr* by using a two-character command macro in place of each command, such as *xx* in place of *xwd -out abc*? (c) rewriting the *plotgraph* program to generate hardcopy directly?

(9) Make a note of the sequence of operations and displays from one or more Automated Teller Machines (ATMs) that you use, and perform cognitive walkthroughs.

(10) Conduct a keystroke-level analysis of the ticket machine of Figure 8.6. This will involve estimating the time taken for some of the special operators involved, for example, inserting money and taking ticket and change. Determine approximate values by timing yourself using a similar machine. From your analysis, work out how many commuter passengers the machine can handle per hour.

Further reading

Card S. K., Moran T. P. and Newell A. (1983). *The Psychology of Human Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates

Chapter 8 provides a full description of the Keystroke-Level Model, with a number of worked examples.

Desurvire H. W., Kondziela J. M. and Atwood M.E. (1992). What is gained and lost when using evaluation methods other than empirical testing. In *People and Computers VII* (Monk A. *et al.*, eds.). Cambridge: Cambridge University Press

A comparative study of several usability analysis methods.