

USABILITY ENGINEERING

Scenario-Based Development of
Human-Computer Interaction

MARY BETH ROSSON

Virginia Polytechnic Institute and State University

JOHN M. CARROLL

Virginia Polytechnic Institute and State University



MORGAN KAUFMANN PUBLISHERS

AN IMPRINT OF ACADEMIC PRESS

A Division of Harcourt, Inc.

SAN FRANCISCO SAN DIEGO NEW YORK BOSTON
LONDON SYDNEY TOKYO

Executive Editor Diane D. Cerra
Publishing Services Manager Scott Norton
Production Editor Howard Severson
Assistant Editor Belinda Breyer
Cover Design Ross Carron Design
Cover Image *Pillar of Education*/© Hans Neleman/Imagebank
Text Design Detta Penna
Illustration Natalie Hill Illustration
Composition TBH Typecast, Inc.
Copyeditor Barbara Kohl
Proofreader Carol Leyba
Indexer Steve Rath
Printer Courier Corporation

Designations used by companies to distinguish their products are often claimed as trademarks or registered trademarks. In all instances in which Morgan Kaufmann Publishers is aware of a claim, the product names appear in initial capital or all capital letters. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

Morgan Kaufmann Publishers
340 Pine Street, Sixth Floor, San Francisco, CA 94104-3205, USA
<http://www.mkp.com>

ACADEMIC PRESS
A Division of Harcourt, Inc.
525 B Street, Suite 1900, San Diego, CA 92101-4495, USA
<http://www.academicpress.com>

Academic Press
Harcourt Place, 32 Jamestown Road, London, NW1 7BY, United Kingdom
<http://www.academicpress.com>

© 2002 by Academic Press
All rights reserved
Printed in the United States of America

06 05 04 03 02 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means—electronic, mechanical, photocopying, or otherwise—without the prior written permission of the publisher.

Library of Congress Control Number: 2001090605
ISBN: 1-55860-712-9

This book is printed on acid-free paper.

5 CHAPTER

Interaction Design

Doors let people into spaces. That's what people are looking for when they come up to a door, whether it's made of wood, metal, glass, or some unknown material: Where is the handle, and how do I use it to unlatch and push open the door? As Don Norman has pointed out in his book *The Design of Everyday Things* (1988), even such an obvious and simple user need is easily defeated by poor design.

Norman (1988, 3–4) illustrates his point with an amusing anecdote of a friend “trapped” in the doorway of a European post office. The door in question was part of an outside entryway (a row of six glass doors), with an identical internal entrance beyond. As the man entered through the leftmost pair of doors, he was briefly distracted and turned around. The rotation caused him to slightly shift his position to the right. When he moved forward and pushed a door in the next row, it didn't move. He assumed it must be locked, so he moved to the next pair of doors. He pushed another door; it also refused to move. Beginning to feel confused, he decided to go outside and try again. But now when he pushed the door leading back outside, it also didn't move. His confusion turned to mild panic . . . he was trapped! Just then a group of people entered at the other end of the doorways. Norman's friend hurried over and followed them, and was successful this time.

The problem was a simple one and would have been easy to avoid. Swinging doors come in pairs, one side containing a supporting pillar and hinge, the other one free to swing. To get through, you must push against the swinging side. For these doors, the designers went for elegance and beauty. Each panel was identical, so there were no visual clues as to which side was movable. When Norman's friend accidentally changed his position, he became out of sync with the “functional” panels within the row of glass. The result was an entryway that looked nice but provided poor support for use.



The goal of **interaction design** is to specify the mechanisms for accessing and manipulating task information. Whereas information design focuses on determining which task objects and actions to show and how to represent them, an interaction design tries to make sure that people can *do the right things at the right*

time. The scope of possible action is broad—for instance, from selecting and opening a spreadsheet, to pressing and holding a mouse button while dragging it, to specifying a range of cells in the spreadsheet.

Interaction design focuses on the Gulf of Execution in Norman's (1986) analysis of human-computer interaction (Figure 5.1). The user begins with a task goal, such as the desire to investigate an irregularity in last month's budget. To pursue this goal, the real-world objective is translated into an appropriate system goal—a computer-based task such as examining an Excel spreadsheet. The system goal is elaborated as an action plan that specifies the steps needed to achieve the system goal: point at an Excel icon, double-click to open it, point at the first cell containing a sum, and so on. Finally, the plan is executed: The mouse is grasped and moved until the cursor is over the icon, a double-click is performed to launch Excel, and the pointer is moved to the bottom of the first column.

The example in the figure continues through the cycle to emphasize the important role of system feedback. While the execution takes place, some visual changes appear; for instance, when the file is opened, a new figure (the window) is seen. These changes are interpreted with respect to the spreadsheet context, and ultimately with respect to the budget question.

This is a deliberately simple example, but even a trivial interaction such as opening a spreadsheet can be undermined by usability problems. Suppose that

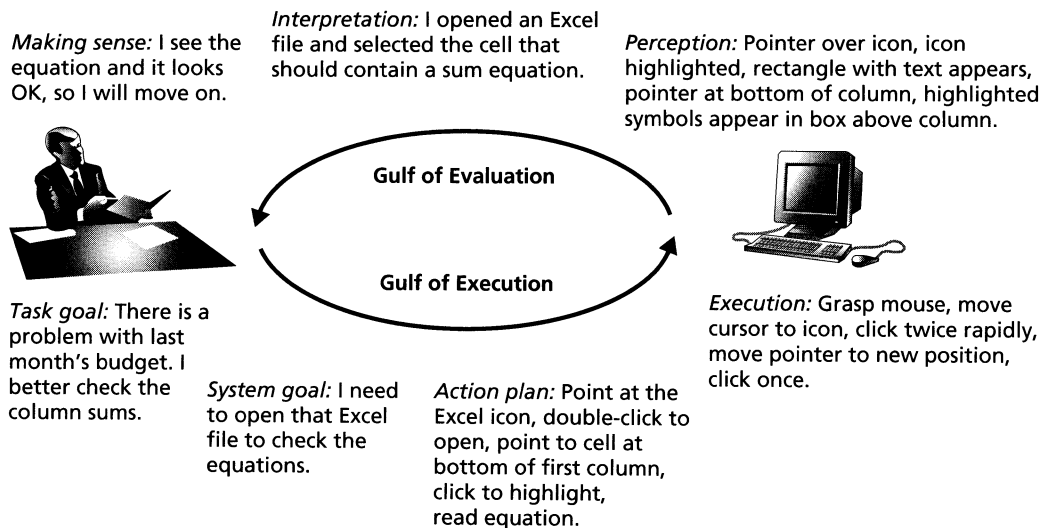


Figure 5.1 Stages of action in a budget problem: choosing, planning, and executing an action, and then perceiving, interpreting, and making sense of the computer's response.

the spreadsheet is very large and only a portion of it shows when it is opened, making it difficult to determine its status; or suppose that it has links to other files that are not present, resulting in warning or error messages. And, of course, opening the spreadsheet is just the beginning. As the task progresses, the computer responds with changing information displays that must be perceived, interpreted, and understood in order to plan and carry out the next set of steps.

As in Chapter 4, we have used Norman's (1986) framework to organize our discussion of interaction design. Here we are concerned with the three stages making up the Gulf of Execution—selecting a system goal, creating an action plan, and executing the action plan. For the sake of simplicity, we limit our discussion to standard user interaction techniques—the **WIMP** user interface style (windows, icons, menus, pointers), the default on most PCs and workstations.

As in all aspects of system development, designers have many options to choose from in designing a user interaction. Their goal is to compose and sequence user-system exchanges in a way that is intuitive, fluid, and pleasant for the task at hand. Doing this depends on understanding the details of the usage situation. There are no simple right or wrong answers; as usual, interaction design is peppered with tradeoffs.

5.1 Selecting a System Goal

To pursue a task with computer support, a user must first translate his or her real-world goal into a software-oriented goal, also known as a **system goal**. The simplest case is one where the system object or action is identical to the real-world concept—perhaps in our example above, the accountant sees an object named “last month’s budget.” This is a very close match to what is wanted; deciding to open it is trivial. The system goal in this case has high **semantic directness**, in that the user’s task goal is mapped very easily onto an appropriate system feature. Of course, the names or pictures of system objects and actions do not usually match task goals exactly, so some amount of processing and inference is required. One goal of interaction design is to minimize this cognitive effort as much as possible.

5.1.1 Interaction Style

A powerful technique for helping people translate their task goals into system goals is **direct manipulation** (Shneiderman 1983). A direct-manipulation user interface is built from objects and actions that are direct analogs of objects and actions in the real world: User interface controls look like buttons that can be

pressed; and data containers look like folders that are grabbed, dragged, or stacked. An active application looks like a window that has been opened for the user to see inside. Choices are shown as menus to be opened and browsed so that items can be selected.

User interface controls that look or sound like familiar objects in the real world simplify the problem of choosing a system goal (Hutchins, Hollan, & Norman 1986). If a user wants to put something away, there are folders waiting to be used. When a user wants to organize information, the objects are on the screen waiting for action. Of course, direct-manipulation techniques require that the right objects and controls are present at the right time—displaying a large set of folders on a screen will be of little help if the goal involves navigation to a Web page.

Even this simple example makes it clear that direct manipulation is not a universal interaction technique (Tradeoff 5.1). Persistent visibility of objects and actions is essential, but a large number of display elements will lead to visual clutter. People must decide which of their many tasks are frequent or important enough to “leave out in the open.”

TRADEOFF 5.1

Visible user interface controls that are analogs to real-world objects simplify the mapping from task to system goals, BUT not all task goals have visual analogs.

Direct manipulation also requires that objects can be represented visually, and that the operations on these visual entities are analogs to physical actions (e.g., pointing, selecting, and dragging). But there are many system concepts that have no obvious visual representation. In the accountant example, how could a system visually represent “the two managers who did not yet turn in their budget numbers”?

Direct-manipulation user interfaces are often complemented with some form of a **command language**. A command language consists of a vocabulary and composition rules (syntax) used to identify and manipulate task objects indirectly. Instead of pointing at a file or a piece of data, a user types or says its name, or specifies it through a logical expression, a mathematical equation, or some other symbolic description. In these cases, the distance from a task goal to a system goal can be substantial—the user must remember and produce the right vocabulary words in the right order.

Expressing system goals with commands is economical and flexible. Text requires minimal display space, and simple commands can often be combined to

create more complex expressions. This makes it possible to satisfy many different system goals with a relatively small vocabulary. But even for small vocabularies, learning the rules for specifying and ordering commands (the command syntax) can be difficult (Tradeoff 5.2). Many common objects and actions have multiple names in natural language (e.g., copy/ duplicate, move/relocate, and table/matrix). If these concepts represent possible system goals, users must remember which of the synonyms to use.

TRADEOFF 5.2

Expressing system goals indirectly with names or symbols is flexible and economical, BUT learning a command vocabulary can be difficult or tedious.

Buttons and menus offer an interesting compromise between direct manipulation and command-based interaction. They are persistent visible objects that users can point at and select. But the content that they represent is usually a command. In a menu system, complex command expressions may be constructed through a sequence of choices. For example, the procedure for opening a new browser in Netscape might be summarized as “Execute the File command New with an argument of Navigator.” Indeed, one reason that menus are so pervasive in WIMP user interfaces is that they have a flexibility and economy similar to command languages, while offering the advantage of recognition over recall.

5.1.2 Opportunistic Goals

Sometimes a person has no particular task goal in mind. For instance, someone first starting up a computer in the morning may have no specific agenda, and instead relies on the computer display to remember what needs doing. In such situations, attractive or convenient system goals may be adopted in an opportunistic fashion.

Opportunistic behavior is evoked by perceptually salient or engaging elements in the user interface display. An interesting possibility is detected, which causes the user to remember a task or to adopt a new goal. A familiar example is the response often exhibited on arrival of new mail, where users drop whatever task they are engaged in to check out a new message. Opportunism is also common when novice users are confused or distracted, and seek guidance from the system about what to do next. In these cases, any object or control that looks intriguing or helpful may be accepted as the right thing to do.

TRADEOFF 5.3

Intriguing task options encourage flexible goal switching, BUT opportunism may lead to inappropriate, confusing, or frustrating experiences.

In most cases, opportunism is not a serious usability problem. Setting aside one task to pursue another can enhance feelings of flexibility and increase the spontaneity of one's activities. However, designers should analyze sources of opportunism in their user interfaces, and seek ways to minimize it when it would interfere with task goals (Tradeoff 5.3). Novices become seriously derailed when they are drawn into complex or exotic functionality (Carroll 1990). People may want to know when new email arrives, but they should be able to deactivate such alerts when concentration is important.

5.2 Planning an Action Sequence

The steps needed to achieve a system goal comprise an **action plan**. With experience, many such plans will be learned and automated, such that they require little conscious thought (Anderson 1983). Most users do not consciously plan the steps for accessing and making a selection from the bookmark list in their Web browser; opening a spreadsheet may well happen without conscious attention. However, for more complex tasks, or for people working with a new application, the user interface is a critical resource in determining what steps to take (Payne 1991).

The concept of a plan is related to the task analysis techniques discussed in Chapter 2. Task analysis specifies the steps and decision rules needed to carry out a task; this can be seen as an idealized action plan for the analyzed task. Plans can be decomposed and analyzed at many levels of detail, depending on the interaction concern in focus (e.g., making a selection from a list box versus constructing a piechart). First-time or occasional users may need to think about the details of selecting or manipulating individual user interface controls, but experienced users will operate at a much higher task-oriented level of abstraction.

Action planning is an active process. People retrieve what they know about a system from their mental models. They use the system information available and make inferences to fill the gaps, often relying on experiences with other systems. As a result, the plan guiding the behavior of any one user may overlap only partially with the action plan intended by the designer. People are not machines. Even if we could somehow be taught every possible plan for every possible contingency, we would be unable (or unwilling!) to ceaselessly retrieve and execute these plans in rote fashion.

5.2.1 Making Actions Obvious

How do users know what to do at all? To a great extent, they learn by experience. Users rely on their current mental models and on their reasoning ability to decide what to do. As execution takes place, system feedback may lead them to revise or elaborate their action plan (i.e., through perception, interpretation, and making sense). The success and failure of such episodes results in learning about what works and what does not work; mental models are updated and plans are reinforced or revised.

One way to help users learn what to do is to make it easy to predict, by trying to emulate real-world tasks (Moran 1983). For example, people editing a report will circle or underline a piece of text, and then write an editing mark or comment near it. A word processor that follows this scheme will be easier to understand than one that expects users to first enter their comment and then point to the text it describes. Thus, one design strategy is to document existing procedures, and then define action plans that build on these procedures. The problem with this is that most software projects seek to enhance or improve current tasks. This means that there will always be computer-based tasks that have no real-world analogs (Tradeoff 5.4).

TRADEOFF 5.4



Action plans that correspond to real-world tasks and manipulations are intuitive and easy to learn, BUT many computer functions extend real-world tasks.

An effective direct-manipulation interface can also simplify action planning. The same physical analogies that aid selection of system goals (recognizing a folder as a place to put things) also help to suggest what actions to take (grab and open the folder). This effect on action planning is related to the concept of affordances discussed in Chapter 4. People need not memorize “press a button to activate it”; a screen button affords pressing because it looks like a real-world button. Dimming choices on a menu makes the grayed-out items look inactive, discouraging inappropriate selections; even a relatively subtle affordance like this can be important in ensuring smooth interaction.

As pointed out earlier, it is impossible to support all user tasks with direct manipulation interfaces. Physical analogies work well for simple actions, such as identification, selection, movement, interconnection, and duplication operations. But how do you carry out a search by direct manipulation or apply a global substitution? Researchers working with programming languages have spent decades exploring direct-manipulation techniques for writing programs, but support

for logic, abstraction, and reuse continue to challenge these efforts (Cypher 1983; Lieberman 2001; Rosson & Seals 2001).

5.2.2 Simplifying Complex Plans

In WIMP user interfaces, people rely on icons, buttons, dialog boxes, or other user interface controls to guide them through action sequences. The user looks at a menu bar, and one set of choices is offered; he or she opens the menu and another set appears. A menu item is selected, and another set of more specific choices is presented via a dialog box. And so on. This simplifies planning, because users only need to know the next step. What would otherwise be learned as the command words and parameters of a command language is implicit in a sequence of menus, or in the input fields, check boxes, and other controls of a dialog box.

Nonetheless, plan complexity is still a major design concern. People are always trying new things, and as applications become more powerful, the usage possibilities become more complex. Problems are likely to arise when users attempt tasks with many steps—for example, many levels of nested menus, several interconnected dialog boxes, or many links leading to a Web page. A long sequence of interdependent actions is hard to keep in mind, and users can lose track of where they are. This can lead to omission or duplication of steps, or other errors (see “Designing for Errors” sidebar).

Studies of human memory have shown that people have a limited capacity to hold information in mind (Miller 1956). We can hold more information if it is possible to **chunk** it—that is, organize several interrelated bits of information into a single unit. Chunking often is a natural consequence of use; the more times certain bits of information occur together, the more likely they are to become a chunk. Information that naturally occurs together may be chunked even without previous exposure. Common examples of information chunking are people’s names, phone numbers, dates, and so on.

User interface controls help to chunk interaction sequences. Figure 5.2 illustrates this for the task of using Microsoft Word to indent a paragraph. From the perspective of the system software, this task requires seven inputs from the user: specification of beginning and end points identifying the text, selection of the `Format` menu, and of the `Paragraph` option within that menu, selection of the `First line` indentation option, typing an indentation amount, and selecting `OK`. But from the user perspective, the plan includes three chunks: paragraph selection, accessing the paragraph settings, and setting the indentation.

Defining the chunks of an action plan is a critical aspect of interaction design, but chunking that is arbitrary or that ignores implicit task boundaries is

1. Specify text selection start

2. Specify text selection end

3. Select Format menu

4. Select Paragraph option

5. Set Special to First Line

6. Type value for First Line

7. Accept new settings

1. Select text

The example in the figure continues through the cycle to emphasize the important role of system feedback. While the execution takes place, some visual changes appear; when the file is opened, a new figure (the window) is seen. These changes are interpreted with respect to the spreadsheet context, and ultimately with respect to the budget question.

2. Open Paragraph settings

3. Set indentation

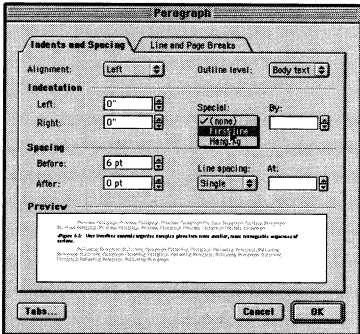


Figure 5.2 User interface controls organize complex plans into smaller, more manageable sequences of actions.

worse than no chunks at all (Tradeoff 5.5). Steps that naturally go together should not be placed in separate chunks. And steps that are very different should not be squeezed into the same chunk. Suppose that in the indentation example above, the line to be indented is identified by pointing at the text (step #5). This would disrupt the third chunk, resulting in a disjointed and awkward interaction.

TRADEOFF 5.5



Decomposing complex plans into chunks aids learning and application of action plans, BUT the sequence may create arbitrary or unnatural step boundaries.

Action planning is also simplified by internal and external consistency (Chapter 4). For example, if some tasks require users to first identify an action and then indicate the object to which it applies, while others require the opposite order, people will almost certainly make errors in learning these procedures.

The memory phenomenon responsible for conflicts of this sort is **interference**. Interference is the inverse of transfer of learning; in these cases prior knowledge leads users to do the *wrong* thing.

5.2.3 Flexibility

People are good at multithreaded activities, that is, pursuing multiple goals at once. We often interrupt ourselves, set aside our current goals, and take on new goals (see Section 5.1.2 on opportunistic goals). This makes us responsive to our environment; we can rearrange task priorities as a function of new information, or even as a function of what seems more or less rewarding at the moment. It also increases our feelings of control—we see ourselves as people who make decisions about and manage our own behavior.

As the power and sophistication of personal computers has increased, multithreaded interaction has become pervasive. Most machines can easily run three or four different applications simultaneously with little or no impact on processing speed. The implications for interaction design are strong: People must keep track of where they are in one plan while they pick up on another; when they return to a deferred plan, they need to remember where they were, so that they can resume. For complex plans with many embedded activities, people will put a plan on hold but expect to maintain the current task context. A user filling out a complex Web order form should be able to leave the form temporarily (e.g., to investigate another product) and return to it later without losing the data already entered.

Multiple overlapping windows are commonly provided to increase the flexibility and control of user interactions. Each window holds the status and

data relevant to an ongoing plan. **Property sheets** are special cases of this general technique; they are opened to investigate or set task-relevant characteristics such as the preferences defined for a Web browser or email program. Users can put aside one task and continue another simply by clicking on a window to bring it (along with its status information) into focus.

An obvious cost of multiple windows is an increase in plan complexity (Tradeoff 5.6). When multiple tasks are underway, people often are forced to take on an extra task—finding and activating windows. They may end up spending valuable time on housekeeping chores such as minimizing, resizing, or rearranging windows. They may also be drawn into tasks that have low priority (opportunism). Providing clear indications of task identity and status (e.g., title bars, the current state of contained data or processes) can help to address this problem.

TRADEOFF 5.6

Allowing plan interruption and resumption enhances feelings of control, BUT management of simultaneous plans is demanding and may increase errors.

A variant of multiple windows is a **tiled display**. This style can now be seen in the many Web applications that use frames. Different categories of information are presented in persistent subareas of the display. An important difference between overlapping and tiled window displays is that users see all of the tiled presentations all the time. In fact, this is a key design consideration: If a task involves multiple related goals and information sets, designing a coordinated tiled display can encourage dynamic construction and switching among plans. Our work on programming tools for Smalltalk demonstrates this—a tiled display supports simultaneous interaction with complementary views of an example application (Carroll, Singer, et al. 1990; Carroll & Rosson 1991).

In order to work on two tasks at once, individual plans must be interruptible. User interaction **modes** work against flexible task-switching and activity management. A mode is a restricted interaction state, where only certain actions are possible. Common examples are an “insert mode” that only accepts text input; an alert box that must be dismissed in order to continue work; or a dialog box whose settings must be accepted or canceled before returning to the main window.

Modes are sometimes necessary—for example, when an urgent system event has taken place and the user must acknowledge this or take some action before continuing. However, in general, designers should avoid putting users into situations where they are forced to complete a plan before continuing. The ever-

present `Cancel` button on dialog boxes is a compromise solution—users may not be able to continue work on their data while a dialog box is open, but at least they can quickly leave the mode.

5.3 Executing an Action Sequence

The final phase of an action cycle is execution of plan steps. In some sense, execution is an inconvenience—what users really want is to accomplish their goals directly, but they must do this by carrying out a sequence of physical actions. On occasion, though, the execution process itself may be rewarding. Video game experts probably feel a sense of accomplishment and reward when they push a joystick just the right amount. In either case, the design of simple and fluid action sequences will greatly impact people's competence and satisfaction in plan execution.

The most important actions to get right are those that are repeated over and over: pervasive actions such as selection, opening, moving, control keys, menu navigation, and so on. Not surprisingly, these are the sorts of interactions addressed by many user interface style guides (Apple Computer 1992; IBM 1991; Sun Microsystems 1990). From a design perspective, pervasive controls are also the elements that developers have least control over; the look and feel of these controls is usually inherited or highly constrained by a windowing system and associated code libraries. Nonetheless, careful examination of these primitive operations can be important in selecting user interface software tools.

5.3.2 Feedback and Undo

One of the most crucial elements of interaction design is **feedback**—the system-generated information that lets users know that their input is being processed or that a result has been produced. If people cannot see how fast they are moving in a space, they cannot adjust their speed to increase accuracy. If they cannot see that a target has been selected, they will not know to manipulate it. If they cannot see what text they have typed, they will not be able to detect and correct mistakes.

The need for feedback is obvious, yet from a software construction perspective, it is easy to ignore: Tracking and reacting to low-level actions require significant testing and code development, so user interface developers may be tempted to minimize their attention to such details. One important responsibility of usability engineers is to make sure that this does not happen.

Of course, constant and complete feedback is an idealization. Every bit of feedback requires computation; input events must be handled, and display updates calculated and rendered (Tradeoff 5.8). As feedback events become more frequent, or as the updates become more complex, system responsiveness will deteriorate. Thus, a challenge for interaction design is determining which aspects of an action sequence are most dependent on feedback, and what level of accuracy is adequate at these points (Johnson 2000).

TRADEOFF 5.8



Immediate and continuing feedback during execution helps to track progress and adjust behavior, BUT frequent or elaborate updates can introduce irritating delays.

An example is window manipulation—early systems animated the movement or resizing of the entire window contents as feedback. However, this made the interactions sluggish. Modern windowing systems demonstrate that a sim-

ple frame is sufficient in most cases to convey size or location changes. (As an exercise, see if you can think of cases where this would not be a good solution.)

Designing task-appropriate feedback requires a careful analysis of a task's physical demands. Speed and accuracy trade off in motor behavior: A task that must be done quickly will be done less accurately (Fitts & Posner 1967). Thus, dynamic feedback is important for a sequence of actions that must be carried out rapidly. Similarly, if accuracy has high priority (e.g., positioning a medical instrument under computer control, or deleting a data archive), extensive and accurate feedback should be provided.

Even with high-quality feedback, execution errors will be made. Frequent action sequences will be overlearned and automated; automated sequences may then intrude on less frequent but similar behaviors. Time and accuracy of pointing depend on target size and distance (**Fitts's Law**; Fitts 1954; Fitts & Peterson 1964). Thus, from a performance perspective, an information design should make objects as large as possible and as close as possible to the current pointer location. But this is not always feasible, and pointing latency and accuracy will suffer. Users also make anticipatory errors—for example, pressing `Delete` before verifying that the right object is selected. And, of course, many execution errors have nothing to do with motor performance, but rather result from distraction or lapses of concentration, as when a user mistakenly presses `Cancel` instead of `Save` at the end of an extensive editing session (see “Designing for Errors” sidebar).

Sometimes execution errors are easy to correct. A mistyped character is easily deleted and replaced with another. In a direct-manipulation system, a mouse that overshoots its target can quickly be adjusted and clicked again. But when errors result in substantial changes to task data, the opportunity to reverse the action—i.e., to undo—is essential. Indeed, this is a key advantage afforded by work within a digital (versus real world) task environment: If the system is designed correctly, we can say “oops, that isn't what I meant to do,” with very little cost in time or energy.

Although some degree of reversibility is needed in interactive systems, many issues arise in the design of undo schemes (Tradeoff 5.9). It is not always possible to anticipate which goal a user wants to reverse—for example, during paragraph editing, do users want to undo the last character typed, the last menu command, or all revisions to the paragraph so far? Another concern is undo history, the length of the sequence that can be reversed. A third is the status of the `Undo` command: Can it also be undone and, if so, how is this interpreted? Most interactive systems support a restricted undo; users can reverse simple events involving data input (i.e., typing or menu choices) but not more significant events

(e.g., saving or copying a file). `Undo` is often paired with `Redo`, a special function provided just for reversing undo events.

TRADEOFF 5.9

Easy reversibility of actions aids confidence and encourages speed, BUT users will come to rely on undo and be frustrated when it “undoes” the wrong thing.

Of course, even simple undo schemes will do the wrong thing at times. In Microsoft Word the `AutoFormat` feature can be used to correct keyboard input as it is typed, such as changing straight quotes to curly quotes. But unbeknownst to most users, these automatic corrections are also added to the undo stack—typing a quote mark causes *two* user actions to be stacked, including the ASCII key code for the straight quote, plus its automatic correction. A request for undo first reverts to a straight quote, a character the user will have never seen while typing!

Feedback and undo are broad issues in user interface design. Although our focus here is on their role in plan execution, feedback contributes to all levels of planning and action. The order summary in an online store helps the buyer make sense of the transaction thus far. Being allowed to go back a step and fix just one problem with the order data will have a big impact on satisfaction. Reminding a user that he or she is about to commit \$450 to the order on submission may be irritating at the time, but it forces the important step of verifying an action with important (perhaps not undoable) consequences in the real world.

5.3.3 Optimizing Performance

An obvious design goal for execution is efficiency. Users asked to input long or clumsy sequences of events will make errors, they will take longer, and they will be unhappy. For routine and frequent interactions, time lost to inefficient action sequences may be estimated and valued in hundreds of thousands of dollars (Gray, John, & Atwood 1992). Thus, it should come as no surprise that much work on user interface design and evaluation techniques has focused on performance optimization (Card, Moran, & Newell 1983).

Perhaps the biggest challenge in optimizing performance is the inherent tradeoff between power and ease of learning. In most cases, a command language is more efficient than a graphical user interface (GUI), simply because users can keep their hands in one position (on the keyboard) and refer indirectly to everything in the system. An experienced UNIX system administrator is a classic image of a power user. In a GUI, users point and click to access objects and actions; this

takes considerable time and effort, especially when objects or actions are deeply nested. However, a GUI is much easier to learn—users recognize rather than recall the available objects and actions, and careful visual design can create affordances to guide goal selection and plan execution.

Most user interfaces support a combination of graphical and text-based interaction. System objects and actions are presented visually as window contents, buttons and menus, icons, and so on. At the same time, frequent functions may be accessed via **keyboard shortcuts**—keyboard equivalents for one or more levels of menu navigation. Particularly for text-intensive activities such as word processing, such shortcuts have substantial impacts on input efficiency and on satisfaction. Comparable techniques using keystroke+mouse combinations can be equally effective in drawing or other graphics-intensive applications. Simple **macros** that chunk and execute frequent action combinations provide a customizable fast-path mechanism.

Whether or not a user interface includes special actions for optimizing performance, careful attention to a sequence of actions can improve execution efficiency. Consider the design of a menu system. The time to make a menu selection depends on where the menu is relative to the selection pointer, how long it takes to reveal the menu, and how long it takes to find and drag the pointer down to the desired item. A design optimized for efficiency would seek to minimize execution time at all these points (while still maintaining accuracy). For example, a context-sensitive, pop-up menu reduces time to point at and open a menu. Dialog boxes that are organized by usage patterns optimize time to interact with the controls. (See Sears [1993] for detailed discussions of layout appropriateness.)

Providing good **defaults** (choices or input suggested by the system) is another valuable technique for optimizing performance. Dialog boxes display the current values of required settings, but if a setting is optional or has not been specified yet, a most-likely value should be offered. It may not always be possible to guess at a good default (e.g., when users enter personal information for the first time), but even if a partial guess can be made (e.g., that they live in the U.S., or that their travel will take place this month), people will appreciate the input assistance, as long as it is not difficult to reset or replace suggestions that do not match task needs. Defaults also help in planning, by suggesting what is normal behavior at this point in a task.

The problem with optimizing an interface for frequent action sequences is that it is difficult to optimize one execution path without interfering with others (Tradeoff 5.10). More conceptual issues arise as well. For example, ordering menu items by frequency of selection may compete with a task-based rationale (e.g., editing operations versus file manipulation). If a frequently used button on

a dialog box is positioned near the starting position of the pointer, the resulting layout may be inconsistent with the visual design program in place. If some menu functions are accessed directly via a cascaded menu tree, while others are accessed by opening and interacting with dialog boxes, inconsistencies in the overall dialog structure can result (e.g., compare *Insert Picture* versus *Insert Object* in Microsoft Word.)

TRADEOFF 5.10

Optimized action paths and good defaults for frequent tasks improve task efficiency, BUT may introduce inconsistencies or intrusion errors into less frequent tasks.

There is no easy solution to these tradeoffs, but working through detailed use scenarios can uncover possible performance issues. For example, the keystrokes of alternate action sequences can be modeled mathematically to compare their efficiency (see, e.g., the keystroke-level model of Card, Moran, & Newell 1980). Current research is aimed at automating this sort of low-level modeling and comparison. For example, Hudson et al. (1999) describe a user interface tool kit in which user input events are recorded by the user interface controls that handle them (e.g., a menu that is opened, or text that is entered into a field). When a usability engineer demonstrates a task, an automatic record of user input can be created and used as the basis of performance modeling.

An important application of performance optimization techniques is users with special needs. A blind user can use a screen reader to hear descriptions of items in a visual display, but careful attention to where and how task objects are displayed can have significant impacts on how long it takes to describe them. Users with motor disabilities can benefit immensely from customization facilities supporting the flexible definition of keyboard macros, speech commands, or other substitutes for tedious pointer-based navigation and selection.

Interaction Storyboards

SBD adapts Wirfs-Brock's notion of **user-system conversations** (Wirfs-Brock 1995) to examine short interaction sequences in depth. Wirfs-Brock uses this technique to elaborate use cases in object-oriented design—a user interaction is modeled as a two-sided conversation, with the user's input events making up one side, and the system response the other. A point-by-point analysis of this sort can help to force out otherwise hidden issues.

Wirfs-Brock generates textual descriptions for each side of the conversation. We used instead a simple sketch of the screen at each point during the dialog, annotated with information about what the user sees and does. The result is a

rough **storyboard**, a graphical event-by-event enactment of a complex or crucial sequence of user-system interactions. (A storyboard is an example of a low-fidelity prototype; see Chapter 6. Note that we have developed this one in a graphical editor to make it more legible, but often a rough sketch would be sufficient.)

Figure 5.7 shows a storyboard developed during the detailed design of the visit scenario. It does not represent the full narrative, but focuses on a portion of the episode that raised special interaction concerns—the actions on miniature windows that either update the main view or open a separate window to the side

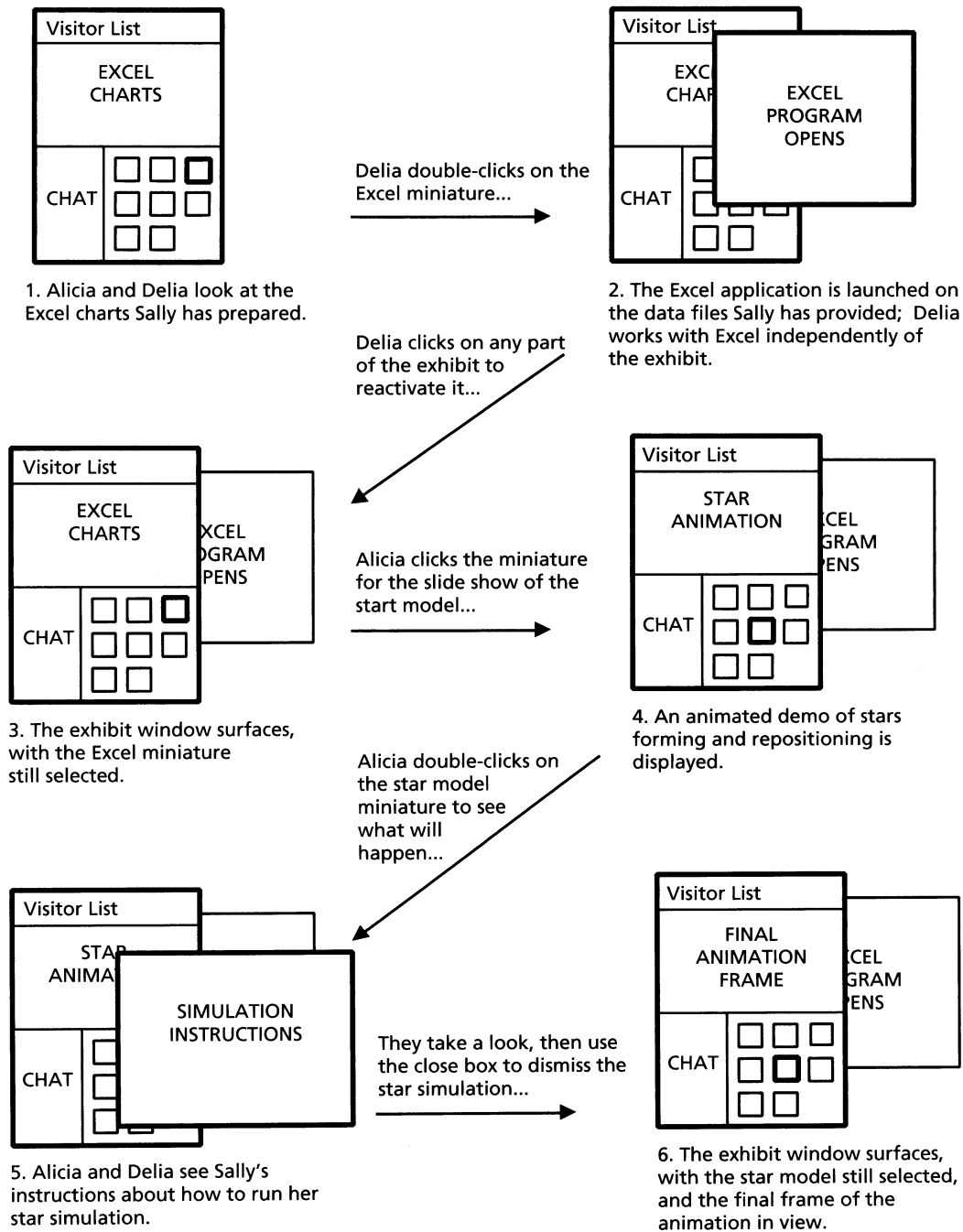


Figure 5.7 A simple storyboard sketching interactions with the miniature windows.

of the exhibit. We took the case where there is already an Excel window open to the side and Alicia and Delia go on to explore the star simulation. This level of interaction seemed reasonable in a visit situation, and we wanted to see if the general scheme of view selection and application launching would work.

In Figure 5.7, the dark borders signify the currently active window and controls. The brief episode shows how Alicia and Delia open first the Excel application, work with it overlaid on the exhibit, and then go back to the exhibit and open another source application. At this point, there are three windows on the display: the overall exhibit view, and the two source applications (in fact, there would also be the original science fair view, which we have ignored in this storyboard for simplicity; it would be a second complex window in the background). The complexity would multiply even more if visitors opened more source applications. By walking through the details of this interaction sequence, we obtained a more concrete feel of what it would be like to click on a control to view its content, and to double-click for more extensive interaction. We persuaded ourselves that the two forms of interaction would be distinct for users (single-click versus double-click) and that they offered a natural mapping to the task goals (the more intense action of double-clicking produces a more intense result of activation).

This storyboarding activity also had a pervasive side effect: We decided that clicking on any window that was part of a set (e.g., an exhibit with several secondary windows open) would cause all of the windows in the set to surface together. The related windows provide a context for interpretation, and so they should be managed as a group.

The virtual science fair envisioned to this point has raised many questions that are best addressed through prototyping and user testing. It is not enough for us to convince ourselves that we have made the right decisions—ultimately, the users will decide. For instance, the techniques just described for exhibit viewing and application launching are in need of empirical evaluation; the creation and access of nested components is another good example. In Chapters 6 and 7 we will show how design scenarios and claims are used in prototyping and usability evaluation.

In the mid-1980s, Digital Equipment Corporation was among the first software companies to define methods for usability engineering. During development of the MicroVMS Workstation Software for the VAXstation I, the usability professionals on the team defined a set of measurable user performance objectives to guide the development process. A central benchmark task was designed in which representative users created, manipulated, and printed the contents of windows. The usability objective was to reduce performance time on the benchmark task by 20% between version 1 and version 2 of the system. As development proceeded, measurements of users' performance on several related subtasks were made to identify areas of greatest usability concern in the design of version 1. The usability problems having the largest impacts on performance were used to prioritize changes and guide the redesign of version 2. In the end the team exceeded their objective, improving performance time on the benchmark task by 37%, while staying within the originally allocated development resources. Interestingly, however, measured user satisfaction for version 2 declined by 25% relative to version 1. (See Good, et al. 1986).



A **usability evaluation** is any analysis or empirical study of the usability of a prototype or system. The goal of the evaluation is to provide feedback in software development, supporting an iterative development process (Carroll & Rosson 1985; Gould & Lewis 1985). Insightful requirements and inspired designs create new possibilities for humans and for their organizations. But there are many ways that goals and plans for new technology can go awry. Despite best efforts and sound practices, the original goals for the system may not in fact be achieved. The system may not be sufficiently useful, it may be too difficult to use or to learn, or using it may not be satisfying. More profoundly, the original project goals may have been successfully achieved, but they may turn out to be the wrong goals. Usability evaluation helps designers recognize that there is a problem, understand the problem and its underlying causes in the software, and plan changes to correct the problem.

Table 7.1 Examples of empirical and analytic usability methods.

Type of Method	Example Methods
<p data-bbox="152 305 390 336"><i>Analytic evaluation:</i></p> <p data-bbox="152 341 528 486">Investigations that involve modeling and analysis of a system's features and their implications for use</p>	<p data-bbox="528 305 1135 372">Claims analysis: system features are analyzed with respect to positive and negative impacts</p> <p data-bbox="528 388 1135 497">Usability inspection: a set of guidelines or an expert's general knowledge is used as a basis for identifying or predicting usability problems</p> <p data-bbox="528 512 1135 657">User models: a representation of the mental structures and activities assumed during use is developed and analyzed for complexity, consistency, and so on.</p>
<p data-bbox="152 699 401 730"><i>Empirical evaluation:</i></p> <p data-bbox="152 735 528 844">Investigations that involve observation or other data collection from system users</p>	<p data-bbox="528 699 1135 766">Controlled experiment: one or more system features are manipulated to see effects on use</p> <p data-bbox="528 782 1135 890">Think-aloud experiment: users think out loud about their goals, plans, and reactions as they work with a system</p> <p data-bbox="528 906 1135 973">Field study: surveys or other types of user feedback are collected from real-world usage settings</p>

7.3 Empirical Methods

The gold standard for usability evaluation is empirical data. Heuristic evaluation produces a list of possible problems, but they are really no more than suggestions. Claims analysis produces a set of design tradeoffs, but the designers must decide whether the tradeoffs are really dilemmas, and if so, which are most problematic. A model-based approach such as GOMS leads to precise predictions about user performance, but has limited application. What usability evaluators really need to know is what happens when people use the system in real situations.

Unfortunately, empirical evaluation is not simple. If we wait to study users until they have the finished system in their workplace, we will maximize the chance for a real disaster. Finding significant problems at that stage means starting over. The whole point of formative evaluation is to support parallel development and evaluation, so as to avoid such disasters. On the other hand, any compromise we make—such as asking users to work with an incomplete prototype in a laboratory—raises issues concerning the validity of the evaluation (i.e., do laboratory tasks adequately represent realistic use?).

The validity of the testing situation is just one problem. Rarely do empirical results point to a single obvious conclusion. A feature that confuses one user in one circumstance might save another user in a different situation. What should we do? What conclusion can we draw? There are technical tools to help manage these problems. For example, we can calculate the mean number of users who experience a problem with some feature, or contrast the proportion who like the feature to the proportion who dislike it. However, the interpretation of descriptive statistics such as these depends very much on the number and characteristics of the users who are studied.

7.3.1 Field Studies

One way to ensure the validity of empirical evaluation is to use **field study** methods, where normal work activities are studied in a normal work environment. As we emphasized in Chapters 2 and 3, people often adapt new technology in un-

expected ways to their existing work practices. The adaptations that they invent and their descriptions of how and why they use the technology in these ways can provide detailed guidance to designers trying to refine a system's basic functions and user interface. Thus, field studies can be valuable in formative evaluation, just as they are in requirements analysis. A field study is often the only way to carry out a **longitudinal study** of a computer system in use, where the emphasis is on effects of the system over an extended period of time.

Suchman's (1987) study of document copier systems is a classic example of a field study. Suchman observed people using a sophisticated photocopier equipped with sensors to track users' actions, and the system offers helpful prompts and feedback. In one example, a person starts to copy a multipage document by placing the first page in the document handler. The copier senses the page and prompts the user to press Start. She does so, and four copies are produced. The user is then prompted to remove the original document from the handler. She does this as well, then waits for further direction. However, the copier next senses the pages in the output tray, and prompts the user to remove them. At this point, the interaction breaks down: The prompt about the output tray does not connect to the user's current goal. She ignores it and instead places a second page into the document handler, triggering a repeat of the earlier prompt to remove the originals. Trying to understand, she misinterprets: "Remove the original—Okay, I've re- . . . , I've moved the original. And put in the second copy."

This simple example vividly shows how a photocopier designer's best efforts to provide helpful and "smart" instructions backfired, actually misleading and confusing the user. As is typical of field studies like this, the details of the episode also suggest design remedies. In this case, providing less help via the prompts led to a better design.

In a comprehensive field study, hundreds of such episodes might be collected. The amount and richness of the data emphasize the key disadvantage of fieldwork—the data obtained has high validity but can be extremely difficult to condense and understand (Tradeoff 7.3). One approach to this is **content analysis**—the evaluator organizes related observations or problems into categories. For instance, one category might be problems with system prompts, another might be button layout, another might be feedback indicators, and so forth. Data reduction of this sort helps to focus later redesign work.

TRADEOFF 7.3

Field studies ensure validity of the usability problems discovered, BUT field study results are extensive, qualitative, and difficult to summarize and interpret.

Field study observations may also be rated for **severity**—each episode is judged with respect to its significance for the user(s). These ratings help to prioritize the data, so that designers can direct most attention to issues of most importance. For example, episodes could be rated on a three-point scale: successful use, inconvenience, and total **breakdown**. While breakdown episodes are clearly most useful in identifying and guiding design changes, it is useful to also include successful episodes in the data sample. Successful episodes may seem uninteresting, but they help to establish a user interaction **baseline** (i.e., what can be expected under normal conditions).

Ethnographic observation attends to actual user experiences in real-world situations. Thus, it addresses many of the concerns about the validity of empirical findings (although the presence of an ethnographer may also influence people's behavior). But this style of work is costly. Collecting field data is very time consuming, and analyzing many pages of notes and observations can be laborious.

A compromise is **retrospective interviews**, where people are asked to recall use episodes that they remember as particularly successful or unsuccessful. This method is based on Flanagan's (1954) original work with **critical incidents**—he asked test pilots to report incidents that stood out in their memory of a recent flight. The critical incidents reported by users should not be considered representative or typical; the point is to identify what seems to be important. However, this makes retrospective interviews an extremely efficient method for producing a collection of problem reports.

Unfortunately, self-reported incidents have their own validity problems. It is well known that people reconstruct their memories of events (Bartlett 1964; Tradeoff 7.4). For example, someone might remember a task goal that makes more sense given the result they obtained. The tendency to reconstruct memories becomes stronger as time elapses, so it is best to gather critical incident reports immediately after an activity. Even then, users are often just mistaken about what happened and what caused what, making their retrospective reports difficult to interpret (Carroll 1990). Self-reported critical incidents should never be taken at face value.

TRADEOFF 7.4

Users often possess valuable insight into their own usability problems, BUT humans often reconstruct rather than recall experiences.

Just as users can contribute to requirements analysis and design, they can participate in evaluation. In field studies and retrospective interviewing, they

participate as actors or reporters. But they can also become analysts themselves through **self-reflection**—a person is asked to interpret his or her own behaviors. In such a situation, the evaluator ends up with two sorts of data—the original episode and the actors' thoughts about what it means. We are currently exploring this approach with a collaborative critical incident tool, in which a community of users and designers post and discuss usage episodes (Neale, et al. 2000).

7.3.2 Usability Testing in a Laboratory

A significant obstacle for field studies is that systems are often not fielded until development is complete. Even if a preliminary field trial is conducted, it may be costly to travel to the site to collect observational data. There are also specific technical reasons for evaluating usability in a laboratory setting: Laboratory studies can be small in scope and scale, and they can be controlled to focus on particular tasks, features, and user consequences. Laboratory studies do not have the overhead of installing or updating a system in a real work site, so they permit rapid cycles of user feedback and prototyping. In fact, laboratory tests can be useful well before any design work has been done—for example, by studying users' performance on comparable systems using standardized **benchmark tasks**.

Because laboratory studies can only simulate real-world usage situations, test validity becomes a major concern (Tradeoff 7.5). For example, it is important that the users be representative—they should be similar to the target users in terms of background, age, ability, and so on (Figure 7.3). A team may inadvertently recruit test participants who know a lot about spreadsheet programs but very little about Web browsers, relative to the intended user population.

TRADEOFF 7.5



Laboratory studies enable focused attention on specific usage concerns, **BUT** the settings observed may be unrepresentative and thus misleading.

It is also important to recognize and interpret differences among individuals. For example, a novice user may like a system with extensive support for direct manipulation, but an expert user may be frustrated and critical. This is the general problem of **variability** in test results; dealing with variability is a challenge for all empirical work. Evaluators can address these concerns by studying a large enough set of users that general patterns emerge, or by carefully documenting user characteristics and restricting their conclusions to people who have these characteristics.

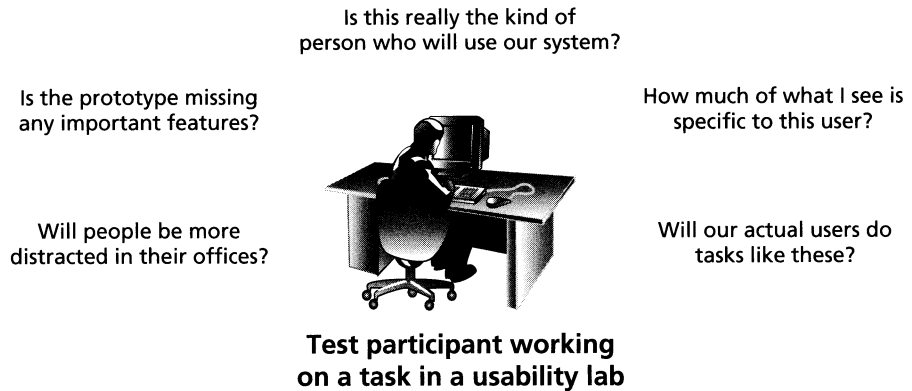


Figure 7.3 Validity concerns that arise in usability testing done in a laboratory.

The prototype or mock-up tested in a laboratory study may also differ from the final system in key respects. The IBM PCjr failed in part because of the rubbery feel of its keyboard. Studies of keyboard prototypes focusing only on the small size of the miniature keyboard would have missed this critical element. The printed graphics pasted onto a cardboard mock-up may have a higher resolution than the resolution real users will experience with cheap displays. What if users working with such displays cannot see the critical cues?

The tasks tested in the laboratory may not be the ones that people will ultimately undertake with the system. Initially, the significance of the World Wide Web was thought to be its improved user interface for transferring files over the network. Spreadsheets were thought to be tools for arithmetic calculation; only later did it become clear that users would also use spreadsheets for planning, reporting, and communication tasks (Nielsen, et al. 1986).

Ironically, another concern for usability testing is the usability laboratory itself! A **usability lab** is a specially constructed observation room that is set up to simulate a work environment (e.g., an office), and instrumented with various data collection devices (e.g., video, one-way observation windows, and screen capture). Users are brought into the lab to perform and comment about carefully constructed test tasks. However, the participants in these studies are insulated from normal work distractions and deprived of many of their daily workplace resources. For example, most work environments involve significant interaction among employees, but this is very difficult to simulate in a laboratory environment.

Sometimes a usability test can approximate a field study. Gould, et al. (1987) describe **storefront testing**, in which a prototype is placed in a semi-public place, such as a hallway near the developers' workroom. Colleagues passing by are invited to try out the prototype and provide feedback. Such user interactions are not examples of real work. The passersby do not have realistic user goals; they are just curious or are trying to be helpful. The usage context may also not be realistic. Gould, et al. were designing a system for a noisy environment, but tested it in the relatively quiet halls of an industrial research laboratory. Nonetheless, a storefront prototype can literally be wheeled out of the developers' laboratory and into a usage setting. The method generates user feedback instantly and supports very rapid prototype refinement and iterative testing.

An important issue for laboratory studies is deciding what data to collect. Most studies gather task performance times, videotapes of user actions, screen displays, and so on. But much of a user's experience is unobservable, taking place inside his or her head as information is interpreted and plans are constructed. Thus, it is also common for usability evaluators to gather **think-aloud protocols**: Users narrate their goals, plans, reactions, and concerns as they work through the test tasks (Ericsson & Simon 1993). The think-aloud protocol can then be analyzed to determine when the person became confused or experienced usage difficulties; the narration before and after a problem often provides insight into the causes and consequences of usability problems.

Usability testing is often conducted in usability labs designed to look like workplace settings (e.g., an office), and evaluators seek to make test participants feel as comfortable as possible. At the same time, it is important to realize that thinking out loud while working is not natural behavior for most computer users! Tracking and narrating mental activity are tasks in and of themselves, and they compete with the application task the user is trying to perform (Tradeoff 7.6). Task performance times and errors are much less meaningful in think-aloud studies. The reporting process also leads users to pay careful attention to their actions and to system responses, which may influence how they plan or execute their tasks.

TRADEOFF 7.6

Externalizing one's goals, plans, and reactions reveals unobservable cognitive sources of usability problems, BUT self-reflection may alter what people do.

Think-aloud studies produce a lot of data, just like field observations. In making sense of the data, evaluators use some of the same techniques they would

apply to field data. For example, they may identify critical incidents directly from the data record (e.g., Mack, Lewis, & Carroll 1983). A more systematic analysis might produce behavior graphs, in which each user action is indicated, along with comments revealing associated mental states (Rosson & Carroll 1996). Because the evaluator controls the requirements of the test tasks, this method can be used to carry out very detailed investigations of how users plan, execute, and make sense of their own behavior.

Almost all usability studies measure time and errors for users performing tasks. To ensure that the times collected are meaningful, evaluators must specify the test tasks precisely enough so that all participants will try to accomplish the same goal. For example, an email task might be “searching for a message received in spring 1997 from Kanazan Lebole that may have mentioned ACM SIGCHI.” Discovering that such a task is difficult or error prone would cause designers to think about how to better support message retrieval in this email system. (Remember that task times and errors collected during think-aloud studies are influenced by the demands of the reporting task.)

Most usability tests also gather users’ **subjective reactions** to a system. Users may be queried in a general fashion (e.g., “What did you [dis]like most?”) or they may be asked to rate the usability of specific tasks or features. An interesting and challenging aspect of user testing is that subjective reactions do not always correspond to performance data. A feature may improve efficiency but also annoy users, or it may slow users down but make them feel more comfortable. For example, early formative evaluation of the Xerox Star revealed that users spent a considerable time adjusting window location—they tried to keep their windows arranged so as to not overlap (Miller & Johnson 1996). As a result, designers decided to not allow overlapping windows. Soon after, however, it became clear that overlapping windows are preferred by most users; this is now the default for most windowing systems. The Star was perhaps so far in front of user and designer experience that it was impossible to make reliable formative inferences from time and error measures.

7.3.3 Controlled Experiments

Most usability evaluation examines performance or satisfaction with the current version of the system or prototype. It tries to answer questions about what is working well or poorly, what parts of the system need attention, and how close developers are to meeting overall usability objectives. On occasion, however, a more controlled study may be used to investigate a specific question. For example, suppose a team needs to understand the performance implications of three

different joystick designs. A controlled experiment can be designed to compare and contrast the devices.

The first step in planning an experiment is to identify the variables that will be manipulated or measured. An **independent variable** is a characteristic that is manipulated to create different experimental conditions. It is very important to think carefully about how each variable will be manipulated or **operationalized**, to form a set of different test conditions. Participants (subjects) are then exposed to these varying conditions to see if different conditions lead to different behavior. In our example, the independent variable is joystick design. The three different designs represent three different **levels** of this variable. Attributes of study participants—for example, degree of experience with video games—could also be measured and incorporated as independent variables.

A **dependent variable** is an experiment outcome; it is chosen to reveal effects of one or more independent variables. In our example, a likely dependent variable is time to carry out a set of navigation tasks. Experimenters often include multiple independent and dependent variables in an experiment, so that they can learn as much as possible. For instance, task complexity might be manipulated as a second independent variable, so that the effects of joystick design can be examined over a broad range of user behavior. Other dependent variables could be performance accuracy or users' subjective reactions. For complex tasks requiring many steps, an evaluator may implement some form of **software logging**, where user input events are captured automatically for later review (Rosson 1983).

Experimenters must specify how a dependent variable will be measured. Some cases are straightforward. Performance time is measured simply by deciding when to start and stop a timer, and choosing a level of timing precision. Other cases are less obvious. If task errors are to be measured, advance planning will be needed to decide what will count as an error. If subjective reactions are being assessed, questionnaires or rating scales must be developed to measure participants' feelings about a system.

The independent and dependent variables of an experiment are logically connected through **hypotheses** that predict what causal effects the independent variables will have on dependent variables. In our example, the experimenter might predict faster performance times for one joystick but no performance differences for the other two. As the number of variables increases, the experimental hypotheses can become quite complex: For example, one joystick might be predicted to improve performance for simple tasks, while a second is predicted to improve complex tasks. Hypothesis testing requires the use of inferential statistics (see Appendix).

Once the experiment variables have been identified, experimenters must choose how participants will be exposed to the different experimental conditions. In a **within-subjects design** (also called repeated measures), the same participants are exposed to all levels of an independent variable. In contrast, a **between-subjects design** uses independent groups of participants for each test condition. In our example, we might have one group of users who work with all three joysticks (within subjects), or we might bring in different groups for each device.

A within-subjects design has the advantage that the variability in data due to differences among users (e.g., some people respond more quickly in general) can be statistically removed (controlled for). This makes it easier to detect effects of the independent variable(s). However, exposure to one level of an independent variable may influence people's reactions to another (Tradeoff 7.7). For example, it is quite possible that experience with one joystick will cause people to learn physical strategies that would influence their success with a second.

TRADEOFF 7.7

Using the same participants in multiple testing conditions helps to control for individual differences, BUT may lead to task ordering or other unwanted effects.

Such concerns are often addressed by a **mixed design**, where some independent variables are manipulated within subjects and others are manipulated between subjects. For example, the different joysticks could be used by different groups of people, but task complexity could be left as a within-subjects variable.

Within-subjects designs are popular and convenient—fewer participants are required, and potentially large effects of individual variability are controlled. Exposing the same participants to two different conditions also allows for direct comparison (e.g., in a series of rating scales). However, experimenters must anticipate nuisance effects, such as task order, and counterbalance the testing conditions as necessary. In complex designs having many independent variables with multiple levels of each, preventing nuisance effects can be challenging.

In any usability evaluation, it is important to recruit subjects who are representative of the target population. But for experimental studies, there are also important questions of how participants are assigned to different conditions, and how many participants will be needed to measure the expected differences among conditions.

The simplest method is **random assignment**: Each participant is placed randomly in a group, with the constraint that groups end up being the same size (or as close as possible to equal sizes; unequal group sizes reduce the sensitivity of statistical tests). Under this strategy, nuisance variables such as age, background,

or general motivation are randomly distributed across the different experimental conditions. Randomization increases the “noisiness” of the data but does not bias the results.

Random assignment is most effective when the number of participants is large; a rule of thumb is to have at least ten individuals in each condition. As the number of participants (the sample, or n) increases, the statistical estimate of random variation is more accurate, creating a more sensitive test of the independent variables. However, this can lead to a dilemma—an experiment with results of borderline significance can often be repeated with a larger number of participants to create a more powerful test, but results may not be worth the extra cost and effort (Tradeoff 7.8). Usability practitioners must carefully balance the needs of their work setting with the lure of reporting a “statistically significant” result. Ironically, a very large experiment may produce statistically significant differences that account for a very small portion of the overall variance in the data. Experimenters should be careful to report not only the statistical tests of differences, but also the proportion of the overall variability accounted for by these differences (see Appendix).

TRADEOFF 7.8

The sensitivity of a statistical test is enhanced by increasing sample size, BUT obtaining a statistically significant result may not be worth the cost and effort of running a very large experiment.

Another assignment strategy is to combine random assignment with control of one or more participant variables. For example, experimenters often randomly assign an equal number of men and women to each group because they worry that gender will influence the results. This helps to ensure that any effects of this particular nuisance variable will be equally distributed across the conditions of interest.

7.4 Science Fair Case Study: Usability Evaluation

Evaluation is central and continuous in SBD. From the first step of activity design, the use context provided by the scenarios serves as an implicit test of the emerging design ideas. The “what-if” reasoning used in analyzing claims expands and generalizes the situations that are envisioned and considered. The focus on design feature impacts is an ongoing source of formative evaluation feedback.

Evaluation becomes more systematic when usability specifications are created from design scenarios and their associated claims (Figure 7.4). The scenarios

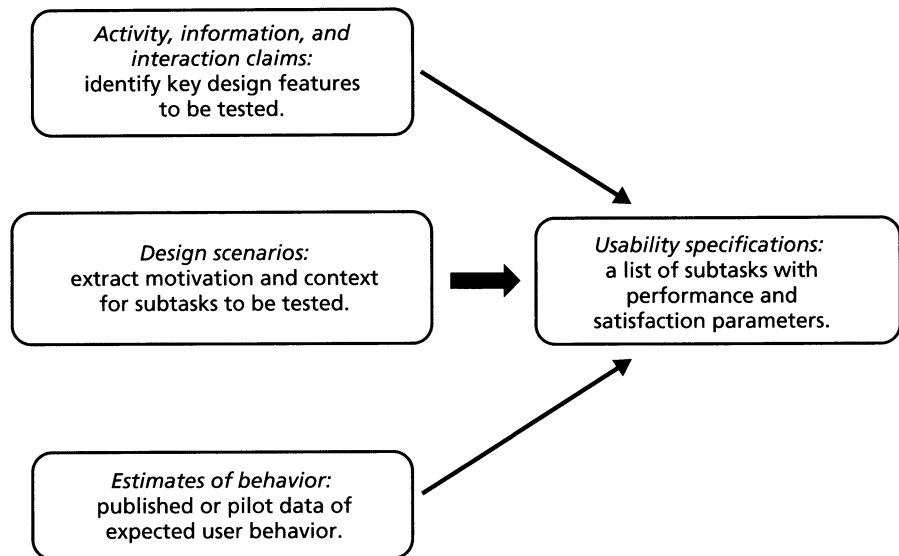


Figure 7.4 Developing usability specifications for formative evaluation.

provide the motivation and activity context for a set of sequential subtasks that will be evaluated repeatedly as benchmarks of the system's usability. The claims analyses are used to identify the subtasks to be tested—recall that claims have been used throughout to track design features with important usability implications. For each subtask, outcome measures of users' performance and satisfaction are defined, creating a testable set of usability specifications (Carroll & Rosson 1985).

The high-level goal of any usability evaluation is to determine to what extent a design is easy or hard to learn and use, and is more or less satisfying. Usability specifications make this high-level goal more precise, transforming it into a set of empirically testable questions. The repeated testing of these tasks ascertains whether the project is meeting its usability goals, and if not, which design features are most in need of attention.

7.4.1 Usability Inspection

As we have seen, analytic evaluation takes place constantly in SBD when writing scenarios and claims. This ongoing analytic work forms a skeleton for empirical studies. Other analytic evaluation methods can be useful as well. For example,

Table 7.3 VSF usability problems identified through heuristic evaluation.

Guideline	Potential VSF Usability Problems
Use simple and natural dialog	Control+F used to synchronize views; Control+I to query activity
Speak the user's language	Young or inexperienced students may not understand "Nested Components"
Minimize user memory load	Chat bubbles stay on the screen only for 20 seconds
Be consistent	People appear as avatars in exhibit space, but as a text list at exhibit; map is replaced by miniaturized windows in exhibit display
Provide feedback	Information on others' activities only available with extra effort; chat bubbles in room overlap for large groups; red color used for alerts will not be detectable by color-blind individuals
Provide clearly marked exits	Relationship between exhibit and nested components not clear; when you change view, what happens to nested component?
Provide shortcuts	Must open each nested component individually, i.e., no "display all"
Provide good error messages	"File type not recognized" doesn't indicate how to fix problem when Excel or other source applications are not installed on client machine
Prevent errors	Multiple independent windows are difficult to distinguish and manage
Include good help and documentation	Help information on how to extend file types assumes familiarity with similar dialogs in Web browsers

we carried out an informal usability inspection using the guidelines suggested by Nielsen (1994); Table 7.3 summarizes usability issues raised during this inspection.

The inspection was carried out as an informal "walk-through" of the design scenarios. We stepped through each scenario and considered whether the actors might have problems in any of the ten areas identified by Nielsen's (1994) guidelines (left column). We did not worry about the severity of the problems at this point, so some of the problems listed in the table may be unlikely or have little impact. For example, few people will have difficulty recognizing that avatars and

text names are different views of the same individuals. However, a visitor forced to carry out many selections and double-clicks to open many nested components may become very frustrated.

A usability inspection such as in Table 7.3 provides an early indication of possible problems, similar to that provided by a claims analysis. Even if no empirical studies were carried out, this list of problems could be prioritized and handed back to the design team for their consideration and possible redesign. Of course, as for any inspection technique, the list of problems identified is meaningful only to the extent the problems will arise in actual use, and without empirical data these judgments can only be based on opinion.

7.4.2 Developing Usability Specifications

In parallel with ongoing inspection and claims analysis, usability specifications were developed for each of the science fair design scenarios. Table 7.4 illustrates the first step in developing usability specifications (we have focused on the two interaction design scenarios presented in Chapter 5; see Figures 5.5 and 5.6).

Each scenario was decomposed into **critical subtasks**—the key features of the system that are influencing people’s experience. The claims analyzed during design were used as a guide in identifying the subtasks. The list is not exhaustive, but rather highlights the system functionality most likely to affect system usefulness, ease of learning or use, or satisfaction. Thus, the subtasks cover a number of the open issues discussed during design (e.g., nested components and use of the control-key commands). This is the sense in which usability specifications support mediated evaluation—the results of analytic evaluation are used to set up an empirical evaluation plan.

Table 7.4 Subtasks analyzed from the VSF design scenarios.

Interaction Design Scenario	Subtasks Identified from Claims Analysis
Mr. King coaches Sally	Identify and synchronize views; Upload local file; Open and work with source application; Create nested component
Alicia and Delia visit the fair	Find location of specified visitor; Join an exhibit; View specified exhibit element; Open and work with source application; Review and modify FAQ; Access and view nested component

Table 7.5 presents two fully elaborated usability specifications. Each scenario has been broken into subtasks, and target levels for users' performance and subjective reactions have been specified. The performance measures are based on time to perform a subtask and the number of errors made. Satisfaction is measured on a 5-point attitude scale. For example, "confusion" after the first subtask is rated on a scale from 1 = "not at all confusing" to 5 = "extremely confusing."

Table 7.5 Detailed usability specifications for two scenario contexts.

Scenario and Subtasks	Worst Case	Planned	Best Case
<i>Interaction Scenario:</i> Mr. King coaches Sally	2.5 on usefulness, ease of use, and satisfaction	4 on usefulness, ease of use, and satisfaction	5 on usefulness, ease of use, and satisfaction
1. Identify Sally's view and synchronize	1 minute, 1 error 3 on confusion	30 seconds, 0 errors 2 on confusion	10 seconds, 0 errors 1 on confusion
2. Upload file from the PC	3 minutes, 2 errors 3 on familiarity	1 minute, 1 error 4 on familiarity	30 seconds, 0 errors 5 on familiarity
3. Open, modify, attempt to save Excel file	2 minutes, 1 error 3 on confidence	1 minute, 0 errors 4.5 on confidence	30 seconds, 0 errors 5 on confidence
4. Create nested exhibit component	5 minutes, 3 errors 3 on complexity	1 minute, 1 error 2 on complexity	30 seconds, 0 errors 1 on complexity
<i>Interaction Scenario:</i> Alicia and Delia visit the fair	3 on usefulness and ease of use	4 on usefulness and ease of use	5 on usefulness and ease of use
5. Find Marge at the VSF	15 seconds, 1 error 3 on awareness	5 seconds, 0 errors 4 on awareness	1 second, 0 errors 5 on awareness
6. Open Sally's exhibit	60 seconds, 1 error 3 on directness	15 seconds, 1 error 4 on directness	5 seconds, 0 errors 5 on directness
7. View data analysis	30 seconds, 2 errors 3 on predictability	15 seconds, 1 error 4.5 on predictability	3 seconds, 0 errors 5 on predictability
8. Open and manipulate Excel charts	5 minutes, 2 errors 3 on engagement	1 minute, 0 errors 4 on engagement	30 seconds, 0 errors 5 on engagement
9. Review and contribute to FAQs	2 minutes, 2 errors 3 on tedium	1 minute, 0 errors 2 on tedium	30 seconds, 0 errors 1 on tedium
10. Access and view Martin's experiment	1.5 minutes, 2 errors 3 on obscurity	45 seconds, 0 errors 2 on obscurity	20 seconds, 0 errors 1 on obscurity

Some scales are written so that a higher number means a more positive rating, while some are reversed. In all cases, the target levels are interpreted as an average across all test participants.

It is important that the usability outcomes in the specification are concrete and testable. Each subtask names a test task that will be observed; the target performance and satisfaction levels specify exactly what measures should be collected. These subtasks are evaluated over and over during development as a benchmark of progress toward agreed usability objectives.

The three levels of outcomes bound the iterative development process: “Best case” is determined by having an expert carry out the task; anything below “worst case” indicates failure. “Planned level” is the actual target, and should be a feasible and realistic statement of usability objectives. Initially, these usability outcomes reflect an educated guess and are based on the design team’s experience with the prototype or with other systems having similar functionality. It is possible that these levels will change slightly as users’ actual performance and reactions are studied, but it is crucial that a team (and its management) take the numbers seriously as targets to be achieved.

Notice that along with time, errors, and satisfaction measures for each subtask, Table 7.5 specifies satisfaction judgments for the interaction design scenario itself. A full scenario includes so many features that it would be difficult to predict precise performance measures. However, the team can certainly ask test users to try out the functionality described in a scenario, and measure subjective reactions to this experience. Measures like these can be used to specify usability outcomes even for a rather open-ended scenario exploration.

7.4.3 Testing Usability Specifications

Usability testing should not be restricted to the design scenarios. Early in development, if a scenario machine is the only available prototype, empirical evaluation may necessarily be limited to these tasks. But when a more general-purpose prototype is ready, new activities and subtasks should be introduced into the evaluation. This is important in SBD, because exclusive attention to a small set of design scenarios can lead to a system that has been optimized for these situations at the expense of others.

Figure 7.5 shows one technique for generating new activity scenarios. The left column summarizes the five design scenarios developed in Chapters 3 through 5. The scenarios on the right were created by bringing in actors with differing backgrounds and motivations, but with overlapping system functionality and user interface features. This strategy works well early in a system’s lifecycle, when only some of a system’s features are available (i.e., that specified in the

Original Scenario	Extension or Generalization
<p><i>Sally plans her exhibit on black holes:</i> An experienced science fair participant organizes her many diverse elements using the template. She pays special attention to components that will make her exhibit more interactive because she knows this will give her points with the judges.</p>	<p><i>Ben and Marissa collaborate on a project:</i> Two students participate in the science fair for the first time. They work independently, and then come together to organize and integrate their sections. Neither is familiar with how to organize or present a science project, so they rely a lot on the templates and help information, and they revise their project a lot as they work.</p>
<p><i>Mr. King coaches Sally:</i> An experienced science fair advisor coaches Sally from home in the evening. He goes over each piece of the exhibit, then helps her make it less complex by finding a way to nest materials.</p>	<p><i>Cheryl makes some suggestions:</i> Cheryl is a retired biochemist who is part of the online seniors group. She sees the VSF advertised in MOOsburg and visits several weeks before the exhibits are done. She browses several biology exhibits under construction and leaves comments on the message boards.</p>
<p><i>Alicia and Delia go to the fair:</i> A busy mother and her daughter log into the fair after school. They see other people there and join an old friend at Sally's exhibit. They see Sally's exhibit, and Delia gets interested and asks questions.</p>	<p><i>Delia brings her friend Stacy back to Sally's exhibit:</i> In school the next day, Delia takes her lab partner Stacy to Sally's exhibit. Sally isn't there, so Delia shows the stored discussion to Stacy, and demonstrates how to use the spreadsheet and the black hole simulation.</p>
<p><i>Ralph judges the high school physics projects:</i> Ralph is an experienced judge with well-developed strategies. He has enough experience with judging and with technology to propose and provide the rationale for a modification to the judges' ratings form.</p>	<p><i>Mark judges for the first time:</i> Because this is his first year, Mark is unsure how to proceed. He spends most of one day on the task, sending out a number of emails for guidance. At certain points he chats online with his peers. Because of his uncertainty, he edits his ratings and comments many times, and prints them out for final review before he submits them.</p>
<p><i>Rachel prepares a summary for Carlisle:</i> The superintendent wants an impressive summary</p>	<p><i>School board member Jenkins thinks about resources:</i> Jim Jenkins thinks the VSF has</p>

(continued)

Figure 7.5 User interaction scenarios form the basis of usability evaluations.

Figure 7.5 (continued)

to use in asking for more science fair resources. Rachel first takes him on a virtual tour, then goes back herself to select and copy out visuals from winning projects.

plenty of resources based on Superintendent Carlisle's presentation. But he goes back to study the site more carefully. He is not very familiar with MOOsburg, so it takes him a while to find and browse just a few exhibits, but he confirms to himself that the fair needs no further support for next year.

design scenarios), but the team wants to evaluate multiple use contexts. Later on, test scenarios representing more radical extensions of the core design (e.g., a teacher who takes her students on a "virtual tour" of the fair) can be developed and evaluated.

The scenarios in Figure 7.5 were used in two sorts of usability testing. Early on, we simply asked test participants to adopt the perspective of an actor (e.g., Sally or Mr. King), and to simulate the scenario activity. For example, a simple introduction such as the following was given:

Imagine that you are Alicia Sampson, owner of a hardware store in Blacksburg. You are already familiar with MOOsburg, but have not visited the Virtual Science Fair. You are busy and somewhat ambivalent about attending science fairs in general, but this year your neighbor Jeff is a participant and your daughter Delia has shown some interest. One afternoon Delia shows you a URL and the two of you decide to log on and visit together. Go to the fair, locate your friend Marge who is already there, join her, and explore the exhibit she is browsing.

The test participants then explored the system with these instructions in mind. We asked the participants to think out loud as they used the system, and we observed their actions with the system. After each scenario we asked them to rate their experience with respect to usefulness, ease of use, and satisfaction (see Table 7.5). The goals of these early tests were very informal—we tried to understand whether the system supported the scenarios as intended, and if not, the major problem areas. In the rest of this section, we describe the more careful testing we conducted on individual subtasks.

Recruiting Test Participants

The participants in usability studies represent the population the system is designed to support. This often means that evaluators will need to recruit indi-

viduals from multiple stakeholder groups. For example, our science fair scenarios include students, parents, teachers, community members, and school administrators as actors. The scenarios also assume certain experience and knowledge (recall the stakeholder profiles in Chapter 2). Sometimes it is difficult to recruit participants from each of these groups (e.g., there are not very many school administrators to draw from). A compromise is to ask individuals from one group to role play the perspective and concerns of another group.

Even when a team can identify representative users, persuading them to participate in a usability session can be challenging. Sometimes a system is novel enough that users will agree to work with it just for the experience. More typically, spending time evaluating a system means taking time away from something else (e.g., work or leisure time). Offering a small stipend will attract some individuals, but ironically, the most appropriate users are often those who are least available—they are busy doing just the tasks your system is designed to enhance! Participatory design addresses some of these problems, because stakeholders are involved in a variety of analysis and design activities. Unfortunately, end users who contribute to design are no longer good representatives of their peers; they are members of the design team.

Regardless of how participants are recruited, it is important to remember that they are just a sample of the entire population of users. Personality, experience, socioeconomic background, or other factors will naturally influence users' behavior and reactions. Gathering relevant background information can demonstrate that a test group is (or is not) a representative sample of the target population. It also aids in interpretation of observed differences among individuals.