

## Chapter 6

# Task Design

Task design is one of the most important aspects of usability testing. You don't just watch users do whatever they want with the interface; you create specific tasks that cover what you want to learn. A good task is like a spotlight that illuminates your interface, showing you the parts that work well and the issues that get in users' way. A good task provides reliable data about key issues so that you get the most from your usability study. But a poor task can mask problems or even turn up false ones. For example, if you ask users to do something that they would never do in real life and they fail, it's difficult to tell whether the interface really has a problem or if you've created an artificial one.

As important as task design is, it isn't easy to do it well. I think task design is probably the hardest part of usability testing, and even after 10 years I'm still learning. This chapter shows you what has worked for me and helps you avoid some common pitfalls.

### *Characteristics of a Good Task*

So what makes a good task? A good task has the following characteristics. (Except as noted, this discussion applies to tasks used for any type of usability testing, not just paper prototypes.)

- ◇ Is based on a goal that matters to the user profile you've chosen.
- ◇ Covers questions important to the success of your product and business.
- ◇ Has appropriate scope—not too broad, not too specific.
- ◇ Has a finite and predictable set of possible solutions.

- ◇ Has a clear end point that the *user* can recognize.
- ◇ Elicits action, not just opinion.

I'm going to discuss the first two characteristics—users' goals and your questions—in detail when I describe the process of creating tasks. For now, suffice it to say that if a task doesn't have the first two characteristics, it is likely to give you misleading information or waste your time.

### **Has Appropriate Scope**

The scope for a task should be large enough for the user to achieve a realistic goal and to answer one or more of your important questions. Typically, tasks of this scope take anywhere from 5 to 30 minutes to complete with a paper prototype.

If you're new to usability testing, your first inclination may be to test one screen at a time. Screens are designed one at a time, but this isn't the best way to test them—you won't uncover the broader issues that arise only when the user sets out to accomplish something. For example, there may be an important function that users need. If you just show them one screen that doesn't contain that function, they may assume that you've provided it elsewhere, when in fact you aren't even aware that it exists. Sometimes your co-workers can review a screen—this is sometimes called “hallway usability”—and spot your more egregious blunders (“Dude, there's no way to delete a record”), but real users have a hard time at this because they're not used to thinking like designers. Unless they've read the functional spec, they don't know all the features that are supposed to be available.

### **Has a Finite and Predictable Set of Possible Solutions**

If you had an entire working interface at your disposal, you might not be concerned about having a finite set of solutions, but when you're using paper you have to make all the prototype pieces to support what the user might do. Thus, you may want to introduce some constraints into your task for the purpose of limiting the amount of preparation you'll need to do.

For example, if you were testing a working version of a gardening site, you might have a task that says, “Last year, raccoons ate the crocus bulbs you planted. Are there any bulbs that don't appeal to critters?” But if there are dozens of bulbs in your online catalog, you might not want to prepare a prototype page for each one. One tactic is to alter the task to constrain the set of possible solutions: “Your

neighbor told you that raccoons won't eat daffodil bulbs—is this true?” That way, you need only the product page(s) for daffodils. Another tactic is to leave the task as is, and if the user tries to look at a bulb page you didn't prepare, simply tell them what it would say about edibility. But in either case, you'd still want to keep a realistic level of complexity in the higher-level pages such as the home page and the category listings; otherwise the task would be artificially easy.

I recommend that there should always be at least one solution to a task. It's best to avoid so-called “red herring” tasks where you tell the user to attempt something you know is impossible—many usability specialists believe that such tasks are not ethical and can result in undue stress on test participants. (Chances are, they'll have a hard enough time with the things the interface does let them do.) If there is a reason you need to use this kind of task, tell the users up front that some tasks may not have solutions and they're allowed to give up.

## **Has a Clear End Point**

It's usually best to avoid overly open-ended tasks such as “Explore this site” or “Find out something interesting about daffodils.” Tasks that lack a clear end point are awkward to facilitate. The users won't be sure whether they are doing the right thing, and it's hard for you to know when to end the task, forcing you to jump in at an arbitrary point and interrupt them.

The users, not you, should decide when the task is done. Sometimes users successfully complete a task but don't realize it because the interface doesn't give them sufficient feedback. It's common for users to decide on their own to verify the results of their work, especially when using an interface for the first time. For example, say they completed all the steps needed to configure the network; chances are, they'll want to ping the devices (send a test message and look for a response) to be sure. If you stop them too soon, you might miss the fact that users need some additional functionality to verify what they've just done. (You should also watch for the opposite problem—users might think they're done, but there's a step they don't know about, or perhaps they've done something incorrectly without realizing it.)

## **Elicits Action, Not Just Opinion**

A good task should cause the user to interact with the interface, not just look at it and tell you what they think. Often, what users say they like does not reliably

indicate what they can successfully use, and vice versa. It's not that users are lying to us—often they are responding to the *concept* of a particular feature, when in reality it may not work the way they envision. For example, several years ago I tested the Function Wizard in Microsoft Excel. One user got stuck trying to create a function that would calculate a mortgage payment, and I finally had to help him. When I asked him what he thought of the Function Wizard, he said, “Now that I know how to use it, I think it's great.” Then I gave him the next task, then the next, and watched him get wrapped around the axle each time. But he never wavered in his professed liking for the Function Wizard.

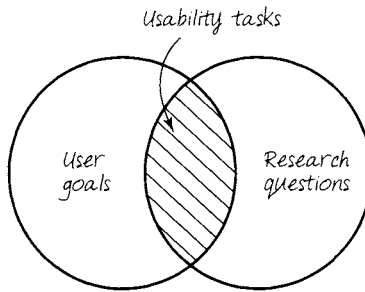
Most of the time during a usability test should be spent watching users work with the interface as opposed to talking about it. If you have opinion-type questions on your list of things you want to know (e.g., “What do you think of this service?”), it's best to save them until the end of the test session, if you even ask them at all. In general, usability testing is not an efficient method of soliciting opinions or other subjective data because you're working with only one or two users at a time rather than the dozens or hundreds of people you'd have with other methods, such as focus groups or surveys. Depending on the completeness of the prototype, opinion questions may be even less useful in paper prototype tests than they are in usability tests of finished products.

I usually avoid opinion questions, although I'll make an exception if someone on the product team has a burning desire to ask one and he or she understands the limitations I've just described. Often, it's pretty clear from the users' comments whether they like or dislike something.

## Overview of the Task Creation Process

In a good task, the user covers the areas of the interface that you have questions about while on their way to accomplishing something that *they* care about. Tasks for usability tests come from the intersection of two lists: your questions about the product and users' goals (Figure 6.1). So let's look at how you get those two lists and then create tasks from them. The steps are as follows:

1. List the users' goals and accomplishments that can be achieved using your interface.
2. List your questions—issues, risks, and other concerns about the interface (you may have done this as part of the kickoff meeting).
3. Prioritize your questions.



**Figure 6.1** Good usability tasks are drawn from the area of overlap—users are doing something they care about, and along the way they’re answering your questions.

4. Pick a user accomplishment that covers one or more important questions and turn it into a task using the template provided in this chapter. Repeat this step until you have a set of tasks that covers your questions.
5. Determine the order of the tasks and whether you’ve got the right amount of work to fill your testing time.
6. Write the instructions that will be given to users.
7. Reality-check your tasks.

As indicated in the previous chapter, task design should be done by the core team, plus others who have a strong interest in what is tested and/or who have knowledge about users and what they do. Although it’s possible for one person to create tasks, the discussion that takes place during the task design session can be quite valuable because it will reveal all sorts of assumptions (often conflicting ones) about what users will do with the interface.

## Step 1: List User Goals

If your interface works flawlessly but doesn’t do anything that users care about, all your hard work will be for naught. So start by making a list of goals and accomplishments that are important to the user profile you’ve chosen for this usability study. What are people using your interface to do? What is important to them? How does your product make their life better? Pay attention to things that users do frequently—if you’re developing an online brokerage and users can’t execute a stock trade, you’re sunk. Also consider goals that are infrequent but really impor-

tant—if users can't use your backup software to *restore* a file, it doesn't matter how easy the backups were to do.

While attempting to list user accomplishments, some product teams realize that they really don't know what their users do. This is not the end of the world, but it indicates the need for research into users and their needs, which is beyond the scope of this book. (But not, fortunately, beyond the scope of other books in the References section.)

Even if you aren't sure what your users' goals are, you can probably make some educated guesses. Another idea is to seek out people in your company who have contact with users (for example, salespeople, trainers, customer service representatives) and get their input. Just keep in mind that it's especially important to reality-check what you come up with, as I describe later.

## Functionality versus Goals

When you're used to thinking about functionality, it can be hard to switch your mind-set to think about the users' goals. You may find yourself listing functionality. When this happens, I keep asking the product team, "Why is that important?" until I hear an answer that contains a user goal.\* The user's goal is often broader than the function the team started with. For example, someone may initially say that a user goal is to buy a stock—the why questions help them realize that the user's true goal is a well-funded retirement (or perhaps the excitement of gambling). Table 6.1 shows more examples for a hypothetical online job database.

## Step 2: List Your Questions

Next, make a list of all the questions and concerns you have about the product. As discussed in the previous chapter, identifying risks and concerns is a helpful thing to do during the kickoff meeting, so you might use that list as your starting point. Some issues might be based on user feedback while others are just a nagging feeling in someone's mind. They are often, but not always, necessarily phrased as questions. Some issues are fairly specific, as illustrated by the following examples compiled from several projects:

---

\* I call this the small-child tactic. Kids have an infinite capacity for asking why. Ironical that part of being a successful consultant involves behaving like a small child.

**Table 6.1** Functionality versus User Goals for a Job Database

<i>Functionality</i>	<i>Why Is This Important?</i>	<i>Rephrased in Terms of User Goal</i>
Register and upload resume	Person wants to find a new or better job. But having a resume online is just one step toward this goal.	Decide whether this service would be useful to me in finding a new job. If it's not going to be worth my time, I won't bother uploading my resume.
Search by geographic area	Commute time is important; many job-seekers don't want to relocate.	Find all the suitable job openings within a reasonable commute from my home so that I don't waste my life driving to and from work.
Sign up for daily email notification	In a tight job market, many job postings are filled quickly.	How can I find out about new job openings before the other guy does so that I can contact the company first?
Navigate to supplemental articles (resume writing, interviewing)	Person wants to increase chances of finding a good job.	I have an interview next Monday. I haven't interviewed for a while—are there particular things I should or shouldn't do?

- ♦ Source/destination—do users understand these terms, or are they too technical?
- ♦ When the free version times out after 30 days, have we broken their expectations? Will they buy it, or just be mad at us?
- ♦ Some people print the preview, which doesn't look very good. How can we get them to print the PDF instead?
- ♦ Do users understand the Reset/Clear buttons in the Watch list?
- ♦ Tech support gets lots of calls about forgotten passwords. How can we fix this online?

Or you might find that your questions are broad or vague:

- ♦ How do people discover new music, such as music by similar but unknown artists? How do they decide what music to buy?
- ♦ Users come to the site for the free materials, but we want them to become buyers. What would help with this? Showing sample pages of new books?

- ◇ We don't know which aspects of account management our customers want to do online. Does anyone really care about account aggregation?
- ◇ We've gotten feedback that the interface is "disjointed." You can end up someplace without clearly knowing how you got there. No one can give us a specific example, though, so we're not sure what this means.

It's not necessary to separate the issues into specific and broad—I've done this to help with the next point I'm going to make. Just brainstorm a list of the things your team is most concerned about, and write them in whatever order they come up. A typical team comes up with a couple dozen issues; you might have even more.

### Step 3: *Prioritize Your Questions*

As a rule, the specific questions are easier to answer, but the broad questions are usually more important to the success of the product. (Specific issues that are costing the company money, like the forgotten password example, may be exceptions.) So guess which questions get made into tasks first? From what I've seen, it's often the specific ones, because they're easier to investigate. But remember what's at stake—the success of the product, perhaps the entire company, may hinge on the broader issues.

*Remember what's at stake—your company.*

So look at your list and ask, "Which of these are most important to the success of the next release?" (Or the company—the best way to phrase this question depends on the nature of the business you're in.) One way to prioritize the issues is to have people vote for the three questions they think are most important. Often, you can't answer all of the top questions in a usability study, but at least you'll be looking at the forest, not just the trees.

Sometimes the broad questions pertain to patterns that will emerge only after you've watched a number of users do various tasks. For example, if your broad questions are "How do users navigate?" and "Why do customers complain that it's too slow?" you can't really create one specific task that will answer them. In this case, make sure that your set of tasks as a whole will let you watch users do the things that you think will yield clues, and eventually you'll start to see the patterns.



## Step 4: Create a Task

This is the step that's hardest to describe in concrete terms. In essence, take your two lists (user goals and your questions) and pick a user goal that covers one or more of your most important questions. Then turn it into a task by asking, "What's a specific example of this?"



For each task, fill out the following template (which is available at [www.paperprototyping.com](http://www.paperprototyping.com)).

<b>Task #:</b> < Task Name >
<i>Goals/output:</i>
<i>Inputs/ assumptions:</i>
<i>Steps:</i>
<i>Time for expert:</i>
<i>Instructions for user:</i>
<i>Notes:</i>

- ◇ **Task name and number.** Give each task a brief descriptive name and a number. The name helps you remember what its purpose is, and the numbers are useful in usability testing because you can ask the observers things such as, "Shall we skip 3 this time and go right to 4?" without discussing the content of the tasks in front of the users.
- ◇ **Goal/outputs.** What will users have accomplished when they're done with the task? Is there a tangible output? How will they know the task is complete? What might the users do to be sure?
- ◇ **Inputs.** List all the information or resources—tangible and intangible—that a user would need to complete this task. Examples include a valid log-in, business policies, physical objects such as a textbook or a credit card, file names, and so on. Real users may have some of this information in their heads—in your usability task you might have to provide this information. For example, a network administrator probably knows the network configuration by heart, but for your task you'd need to create a network schematic with relevant details, such as server names and IP addresses.

For the inputs, outputs, and everything that happens along the way, you'll need to have consistent data that reflects a realistic scenario. For instance, if users enter information on one screen that appears in a report later, the report should match what users entered to avoid confusion (I talk more about creating data in the next chapter). During task design, start thinking about realistic data that you could use for the task.

- ◇ **Assumptions.** Assumptions are the conditions and prerequisites that are in place at the start of the task. The assumptions depend on what you want to learn from the task. For example, if a task explores how users recover from an error caused by a duplicate record, your assumptions include the condition(s) that cause the error to occur, such as, "An employee with the same name already exists in the database." Often, your assumptions will emerge when you start doing walkthroughs, so you don't have to nail them all down in the task design session—you can add more later. In practice, I sometimes combine assumptions with inputs because the distinction between them isn't crucial; they're both things that we need to consider while preparing the prototype.
- ◇ **Steps.** Write down the steps you expect the user will go through in completing the task. This helps you identify the prototype pieces that you'll need to create. Writing down the expected steps can also be helpful if there will be observers who aren't as familiar with the interface as you are.

Keep the steps mostly at a screen level—no need to list every field on the order form, just say "order form." Don't worry about exact terminology because these steps are just for your benefit in preparing the interface. The user won't see these steps, only the task instructions that you'll write later. Some tasks have multiple paths that lead to success, so jot down any variations, such as "Search OR navigate to lawn & garden page." Put optional steps in parentheses, such as (Review privacy policy).
- ◇ **Time estimate for expert.** Estimate how long it would take an expert (someone on the core team) to complete the task. Ignore any time needed for the system to do its processing and focus on the time spent entering data and clicking buttons. Some tasks, such as composing an email, require time for thinking or creative effort, so allow time for that. In the next step of the task design process, I'll explain how to use this estimate to decide when you've created enough usability tasks.
- ◇ **Instructions for users.** Don't write the instructions for the users when you're filling in the rest of the template. Although task design works well as a group activity, writing the instructions can be done by one person after you've drafted your set of tasks. The following examples include the instructions, but keep in mind that they were written after everything else in the template.

- ◇ **Notes.** The notes section might have several types of information, including the reasons why you created the task, how you'll conduct it, specific things to watch for, and questions to ask users after the task is complete. Information to include in the notes varies depending on what's being tested. Write down whatever information you think will be useful to have on hand during the usability tests. Give copies of the completed task templates to usability test observers because this information helps them get more benefit from watching the tests. It's also quite helpful to the test facilitator.

Because task design is a group activity, you may want to draft each task on a piece of flip chart paper or a whiteboard. Ask someone at the meeting to fill out the template on a laptop as you go so that you'll also have it in electronic form.

## Examples of Completed Task Templates

The following are examples of completed task templates that I've used in paper prototype usability tests. Although these tasks came from very different interfaces, you'll note the following things they have in common:

- ◇ The steps are described at a fairly high level, often tersely. These are intended for the development team, who knows what they mean.
- ◇ Each task would take an expert only a few minutes to complete.
- ◇ The length of the instructions given to users varies depending on the amount of background information the user needs to understand and complete the task. (Note: I have not yet explained how to write task instructions; that's coming in step 6.)
- ◇ For some tasks, there are detailed notes, but for others there are none.

Although three of the four examples happen to be from Web applications, tasks for other types of interfaces look much the same. In fact, when you are thinking in terms of goals and instructions for the users, the type of underlying technology shouldn't matter much, if at all. It comes into play only when you're looking at the steps, and to some extent the inputs and outputs. When the goal and task instructions are free of any references to technology or functionality, it's usually an indication that you've been successful at keeping your focus on the users rather than getting bogged down in implementation details.

*Example 1: Web Application for Online College Courses*

**Background:** This application is used by college students. When they purchase a textbook, students get access to the online version of the book and additional course content provided by the instructor. For a new book, this access comes in the form of a card containing a code. This task explores what happens in the case of a used book. Because the code can't be reused, the student must pay separately for access to the online materials.

<b>Task 2: Used Book</b>	
<b>Goal/output:</b>	Successful purchase of online access, resulting in ability to see course assignments
<b>Inputs:</b>	<ul style="list-style-type: none"> <li>◇ Used book with expired access code card</li> <li>◇ Credit card #</li> <li>◇ Course ID as it might be given by instructor</li> <li>◇ QuickStart Guide?</li> <li>◇ Confirmation email</li> </ul>
<b>Assumptions:</b>	<ul style="list-style-type: none"> <li>◇ Student has previously registered but is not logged in</li> <li>◇ Starting from where task 1 left off</li> </ul>
<b>Steps:</b>	<ul style="list-style-type: none"> <li>◇ Complete the registration process (several steps)</li> <li>◇ Purchase online course access</li> <li>◇ Wait for email</li> <li>◇ Log in</li> <li>◇ View course</li> </ul>
<b>Time for expert:</b> 7 minutes	
<b>Instructions for user:</b>	<b>Oral instructions: “For another class, you bought this book used at the bookstore. It came with this card, but this strip has been torn off. Find out whether you can still use this card as you did with the first course.”</b>
<b>Notes:</b>	<ul style="list-style-type: none"> <li>◇ Task 2 tests student registration and course enrollment as in task 1, but this time the student purchased a used book for another class and needs to use the purchase option.</li> <li>◇ Let them try entering the access code and get “already been used” error.</li> <li>◇ Note whether they understand what they’re purchasing.</li> </ul>

*Example 2: Network Administrator Software*

**Background:** This software application is used by network administrators to support the policies and practices within their organization, for example, controlling access to certain types of resources, prioritizing network traffic, and so on. In this scenario, the network administrator works at a college and the goal is to prevent students from using network bandwidth to download music.

<b>Task 1: Block Napster</b>	
<b>Goal/output:</b>	No one on the network is able to access Napster
<b>Inputs/assumptions:</b>	<ul style="list-style-type: none"> <li>◇ Network schematic with IP addresses, switches</li> <li>◇ Switches consist of 1 Cajun and 1 Cisco</li> </ul>
<b>Steps:</b>	<ul style="list-style-type: none"> <li>◇ Log in</li> <li>◇ Set up network</li> <li>◇ Define (name) port</li> <li>◇ Create application</li> <li>◇ Set up policy</li> <li>◇ Make a rule</li> <li>◇ Create domain</li> <li>◇ Create devices—Cisco, Cajun</li> <li>◇ Assign devices to domain</li> <li>◇ Deploy</li> <li>◇ Verify</li> </ul>
<b>Time for expert:</b>	5 minutes
<b>Instructions for user:</b>	You're a network administrator at Whatsamatta U. The college president wants to prevent students from downloading music with Napster. You've just installed [the software] and you want to set it up to prevent anyone on your network from using Napster.
<b>Notes:</b>	

*Example 3: Web Application for Purchase/Download of Music*

**Background:** Strangely enough, this example also deals with college students and downloading music, except in this case it's the desired activity. The users are college students who own a portable MP3 player and are in search of some good

tunes to put on it. In addition to testing the mechanics of downloading music to the PC and MP3 player, in this task we also wanted to see how people went about finding and selecting music.

<b>Task 2: Get 10 Tracks on Device</b>	
<b>Goal/output:</b>	Portable MP3 player has 2 tracks on it that the users like
<b>Inputs/ assumptions:</b>	<ul style="list-style-type: none"> <li>♦ MP3 player</li> <li>♦ Has confirmation email as a result of task 1</li> <li>♦ Each user chooses 1 track according to their tastes</li> </ul>
<b>Steps:</b>	<ul style="list-style-type: none"> <li>♦ Go to [site].com</li> <li>♦ Get to music application (from Download page)</li> <li>♦ Search or browse for first track/artist</li> <li>♦ Download to device</li> <li>♦ Detect hardware, install</li> <li>♦ Search or browse for second track/artist</li> <li>♦ Download to device is grayed out</li> <li>♦ Search or browse for third track/artist</li> <li>♦ Download to device</li> </ul>
<b>Time for expert:</b>	5 minutes
<b>Instructions for user:</b>	<b>Choose 10 songs you like and put them on your portable MP3 player.</b>
<b>Notes:</b>	<ul style="list-style-type: none"> <li>♦ We'll ask the users what music they're interested in and how they'd search/browse to find it. Our prototype probably doesn't have exactly what they'd want, so at that point we'll have them pick something they recognize.</li> <li>♦ If the users get "no search results," do they understand the possible causes?</li> <li>♦ The first track they select will be downloadable to device, but the second track they pick will not be (to see whether they understand why).</li> <li>♦ We won't actually have them download 10 tracks—after the second successful download we'll ask them if they'd use this same process if they were downloading 10.</li> <li>♦ Do not explicitly ask the users about the playlist at this point; the next task is intended to encourage them to discover it</li> <li>♦ Do they realize why some music isn't available (because of legal restrictions)?</li> </ul>

*Example 4: Web Application for Corporate Buyers*

**Background:** The target audience for this application is people who work in purchasing departments at large companies. The application is intended to support the process of selecting vendors according to business rules that the company specifies; for example, vendors will be selected based on quality first, price second.

<b>Task 1: Create RFQ</b>	
<b>Goal/output:</b>	RFQ sent out to potential vendors
<b>Inputs/assumptions:</b>	<ul style="list-style-type: none"> <li>◇ Approved suppliers exist</li> <li>◇ Product catalog exists (we may give them codes)</li> <li>◇ All simple items, no compound ones</li> <li>◇ Desired business rules are known</li> <li>◇ Attachment exists for terms and conditions (t&amp;c)</li> </ul>
<b>Steps:</b>	<ul style="list-style-type: none"> <li>◇ Create high-level RFQ (name, code, type, parameters, policy, suppliers, notes for suppliers)</li> <li>◇ Create requisitions within quotation (name, code, description)</li> <li>◇ Assign items to reqs (select type—simple in this case—category, product, define unique name &amp; code, qty, max, min, price, date)</li> <li>◇ Add attachments for t&amp;c</li> <li>◇ Print, get approvals</li> <li>◇ Submit (define schedule info)</li> </ul>
<b>Time for expert:</b>	10 minutes
<b>Instructions for user:</b>	<p>Welcome to your first day here at Consolidated Monopoly, the world's largest manufacturer of widgets. As an experienced corporate buyer, you'll be able to help us get some equipment we need.</p> <p>We need some desktop computers for our rapidly expanding development organization. We're hoping to get a better deal than the \$800 apiece we've been paying at a local retailer. We need to have 500 PCs delivered in about a month.</p> <p>It doesn't matter whether all 500 PCs come from one vendor or not, and it's okay if we get the monitors from a different vendor than the CPU. The machines should include the latest versions of Windows, Microsoft Office, and Norton Antivirus.</p>

There's a document on your hard disk called "terms&con.doc" that outlines our standard terms and conditions. Make sure that potential vendors receive this document before submitting a bid.

*Notes:*

## **Step 5:** *Number and Order the Tasks*

So how many tasks should you create? Enough to fill your allotted test time, plus one or two extra. Keep in mind that you won't have all the test time to spend on the tasks—it typically takes 5 to 10 minutes to get the user comfortable, briefed, and introduced, perhaps longer if you interview the user about his or her background or work. You may want to reserve another 10 minutes at the end of the test for questionnaires or to discuss interesting issues that arose. So the time available for tasks is typically the test length minus 10 to 20 minutes.

### **The Time Expansion Factor**

Estimating task times is an art, not a science. Although I've gotten better at it over the years, occasionally I'm still way off. But there's a rule of thumb that I've found useful: Take that "time for expert" estimate and multiply it by a time expansion factor. This factor is a crude means of accounting for all the things that cause people in a usability test to work more slowly than an expert under ideal conditions: they're unfamiliar with the interface, they're verbalizing their thought process, they might have to wait for the Computer, and so on.

For software, I multiply the expert's time by a factor of 5 to 10 (I usually use a range, not a single number). In other words, a task that takes an expert 1 minute under ideal conditions will probably consume 5 to 10 minutes of test time. For Web sites, I've found that the time expansion factor seems to be smaller, more like 3 to 5 minutes. (I think this may be because the controls are usually simpler than for software—users are primarily clicking—but this is just a guess.) In looking at your interface, think about how much data entry is involved. Writing by hand takes time, so lots of data entry implies a bigger expansion factor. Also consider how many new or complex concepts are involved, which will affect the amount of time users need to think and/or experiment. Of course, these numbers are only rough guidelines, and your mileage will vary.



Add up the expanded task times and see whether you've filled your allotted test time. It's a good idea to create an extra task or two. Especially in a paper prototype, it's common for users in later tests to complete more tasks because your improvements to the interface make it easier to use.

Make sure that your tasks aren't too long. I've found that a task longer than 30 or 40 minutes can be fatiguing for users (not to mention observers). But there's no hard and fast definition of "too long." If you're testing a lengthy process, you might want to break it down into smaller tasks. Or the facilitator can plan to intervene after a certain amount of time, ask the users to summarize what they've done so far, and offer them a break.

Also be on the lookout for tasks that are very short—5 minutes or less. Sometimes this indicates that you're testing just one screen or function rather than a user goal. Although a short task is not necessarily bad, ask yourself if it makes sense to combine it with another task.

## Dropping and Reordering Tasks

For some types of usability testing it's important to use the same tasks in the same order for each user. This is typically the case if the purpose of the testing is to collect success rates, task times, error rates, or other metrics. But metrics usually aren't the focus of paper prototype tests because the interface is continually evolving. Thus, it's usually not important to use the same tasks in the same order for each usability test—it's okay to vary them.

The purpose of paper prototyping is to provide answers to the questions you're most concerned about. As you conduct usability tests, you'll collect lots of data, and this data can cause you to be more (or less) worried about a particular issue. If you find your concerns changing during a usability study, it's appropriate to change the tasks accordingly. Here are some reasons why you might decide to drop a task and replace it with one of the extras you created:

- ◇ **You're not learning anything new.** Sometimes you learn all you need from a task by watching just the first few users complete it, especially if the interface works well (which does happen, although not as often as we might like). Once a task stops yielding new insights, it's done its job and you can send it into early retirement. Just be careful about reaching this conclusion prematurely—Users A, B, and C might sail past an issue that flummoxes User D. If there's any doubt, keep the task and gather more data.

- ◇ **You've identified a problem but aren't prepared to fix it yet.** Paper prototyping allows you to make many improvements to the interface in a short time, but sometimes you uncover problems that require more in-depth thought. If a task reveals an issue that you all agree needs to be fixed, it's not always necessary to keep banging your head against that particular wall. (However, if you are still trying to understand the nature of the problem, then you should probably keep the task.)
- ◇ **The task is too unrealistic.** If users indicate they'd never do this task, it might not be not worth keeping. But first reexamine the issues that caused you to create the task. If you had a wrong assumption about what users needed, drop the task. On the other hand, if your underlying questions are still valid, perhaps you can fix the task to make it more meaningful to users.

## Step 6: *Write Instructions for Users*

The next step is to write the task instructions you'll give to users. Here are some guidelines.

### **Describe the Goal But Not the Steps**

The purpose of usability testing is to learn how users approach the interface and what naturally makes sense to them. If the task instructions give them too much help, in essence you're training them, and this can mask usability problems. If you're a tech writer, you may have an especially hard time avoiding this trap because you earn your living by writing detailed instructions—it takes a conscious effort not to do this. Instead, the task instructions should explain *what* the user is trying to accomplish, but not *how*. See Table 6.2 for an example.

The exception to the rule of not providing steps is when there's an aspect of the interface that's out of your control or otherwise not interesting for you to test. In that case (which doesn't happen very often), it's okay to explain how to do something. As an alternative to doing this in the task instructions, the test facilitator can step in and help the users past this point.

**Table 6.2** Task Instructions: Bad and Good Example

<i>Bad (Explains How To Do It)</i>	<i>Good (Describes Only the Goal)</i>
<ol style="list-style-type: none"> <li>1. Start Excel</li> <li>2. Go to Tools → MATLAB functions</li> <li>3. Create a new function—from the list of installed MATLAB Excel builder components, load “myclass” from “my component.”</li> <li>4. Select “myprimes” from the list of available methods. Set up inputs/outputs for this function as follows . . . (The original task went on for 2 pages.)</li> </ol>	<p>You have a work colleague who created an Excel Add-In called the MATLAB Function Wizard that allows you to access certain MATLAB functionality. He’s asked you to test it to find all the prime numbers between 0 and 25.</p> <p>(Note: The users were familiar with Excel and the product team wasn’t testing Excel itself, so the explicit wording of “Excel Add-In” is appropriate.)</p>

## Avoid Wording That Reveals the Method

After all those endless meetings where you haggled over the exact wording for a menu option, you shouldn’t use that wording in your task instructions because doing so could provide users with unintended clues. If you ask users to “create a graph” and you have a Create menu with a Graph option in it, you’ve made the task artificially easy. Use synonyms or a picture instead. (Chapter 13 has more examples of ways in which task instructions can cause a bias.)

## A Picture Is Worth 1000 Words

Sometimes visual aids come in handy because they let you avoid using specific nouns and verbs—show the user a graph and say something like, “You want to make one of these for this month’s sales results.” Figure 6.2 shows an example of a picture we used in a usability study of an e-commerce site.



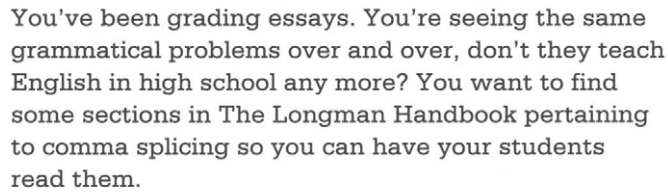
**Figure 6.2** This type of coffee maker is called a French press, but we didn't want to reveal that to users because it might bias them toward using the site's search engine—coffee makers were easy to find via the search engine but hard to find using the links. So we showed them this picture and said, "Your friend has one of these and it makes great coffee, so you'd like to buy one."

## Written, Not Oral

I usually give usability tasks to users in written format so that they can refer back to them as needed. This is important if the user needs to enter data from the task instructions, such as a server name. But there are exceptions to this rule. Sometimes for very simple tasks you can give the instructions orally. Oral instructions are also helpful when you don't want to reveal the spelling of a term, such as, "See if there are any new treatments for osteoporosis." But when in doubt, use written task instructions.

## One Task Per Page

Print each task on a separate page and give them to the users one at a time. This gives you more flexibility to skip or reorder tasks depending on what happens in the usability test. It also avoids making users feel bad if they don't complete all the tasks (especially if you have a couple of extra tasks in reserve). I like to use  $5 \times 8$  index cards for tasks, but regular paper works fine too. See Figure 6.3 for an example. Have an extra copy on hand in case the user jots something down on the task page.



You've been grading essays. You're seeing the same grammatical problems over and over, don't they teach English in high school any more? You want to find some sections in The Longman Handbook pertaining to comma splicing so you can have your students read them.

**Figure 6.3** An example of a task used in a study with college-level English teachers. It contains a subtle joke—can you see it?

## Use Humor Carefully, If At All

Write your tasks in a professional but friendly tone. Most usability specialists agree that it's best to avoid overly clever or “cutesy” tasks. Although humor is a good thing, using it in usability testing can backfire if it accidentally strikes too close to home or the user feels insulted. I once heard of a task that read, “You've just been laid off—ha ha! See what outplacement services the HR site offers.” Unfortunately, one of the users had just experienced a painful layoff and started crying. So that's definitely an example of what not to do!

On the other hand, although paper prototyping is a serious activity, it isn't necessarily a solemn one. I see nothing wrong with using subtle or lighthearted humor to help establish a relaxed atmosphere in the test setting, provided that it's appropriate for your audience. The task shown in Figure 6.3 was used in a usability study with college-level English teachers. (The joke, in case you didn't catch it, is that the second sentence contains a comma splice. To an English teacher, this is funny.) I've also seen developers draw elaborate hourglass cursors to use when the Computer needs time to find the next screen—these can make users smile. Props, such as the one shown in Figure 6.4, can also be humorous. But the bottom line for humor is to carefully consider its effect on your users and err on the conservative side.



**Figure 6.4** We didn't really need this paper "credit card" for the e-commerce site we were testing, but it made people laugh.

## **Step 7:** *Reality-Check Your Tasks*

Sometimes product teams lack a clear understanding of how and why users will use the interface. Although in an ideal world we'd have this information, sometimes we don't and we have to guess. Creating tasks from made-up user goals isn't inherently evil as long as you don't confuse them with reality. Perform a "reality check" by asking users at the end of the task if it's something they'd actually do and/or care about. If they say no, modify the task or omit it. Otherwise, you risk getting meaningless—or even misleading—feedback.

### **From the Field: When the Reality Check Bounces— A Cautionary Tale**

I worked on a medical project where we tested a software prototype with radiologists. We were having a pretty high success rate with task completion. The development team was very happy with the way things were going, but I had an uneasy feeling about the results. I couldn't pin down why; it was just my subjective sense of how the users were responding.

In one of the last tests, the test participant completed the task without any problem but didn't really seem satisfied and was not terribly responsive to the follow-up questions. I finally asked him if this was how he would want to do the work described in the task. Well, the flood gates opened. "Oh no," he said, and proceeded to give a very clear explanation of what he'd actually like to do and why.

### **From the Field: When the Reality Check Bounces— A Cautionary Tale—cont'd**

Fortunately, the development team was observing the test. It was clear to everyone at that point that the task was fundamentally flawed because real users would never do it the way we'd envisioned. This was a very important finding and we wished we'd learned it earlier.

*Contributed by Cay Lodine*

If you're really uncertain about the user goal that a task is based on, you might want to do the reality check before you use the task. Perhaps the people in your company who have contact with users can tell you if the task bears any resemblance to the reality they've seen. Maybe they can even put you in touch with some users so you can ask them yourself.

The process of task design can open up a whole new world of questions about your users and what they do. But don't be discouraged if you feel like you're raising questions faster than you can answer them—this is actually a good sign because it indicates that you've begun to think in a user-centered way.

Just like designing a usable interface, creating good tasks is a complex and iterative process. The internal walkthroughs described in the next chapter will help you refine your tasks as well as create your interface.