Chapter 7

Preparing the Prototype

This chapter explains how to create a paper prototype and prepare for usability testing by doing walkthroughs. The prototype creation and walkthrough activities happen iteratively—make some prototype pieces, do a walkthrough, figure out what's wrong or missing, and repeat. There are no hard and fast rules for how long to spend on each iteration, but as noted in Chapter 5 this process typically takes a total of 2 or 3 days, perhaps spread out over a couple of weeks.

List the Pieces Needed for the Tasks

Make a list on a whiteboard or flip chart of all the screens your prototype will need to support the tasks you've created—the steps from your completed task templates are a good starting point. Keep the list fairly high-level; you don't need to specify every menu or drop-down list because the person making a screen is responsible for its details. For example, the task of opening a retirement account at a financial Web site might require the following screens:

- Browser window with controls
- Home page
- Existing user log-in
- Create new account screen(s)
- Regular/Roth IRA explanation
- Account summary

- ◈ Privacy policy
- ◈ Terms and conditions
- ◈ Retirement savings calculator
- ◈ Mutual fund screener

The list doesn't have to be in any particular order because it's just for your benefit in identifying what to prepare. It's usually fastest to create this list *without* looking at the existing interface, if there is one; otherwise you risk digressing into a discussion of the current design.

Don't Forget the Data

There is likely to be some data associated with the tasks, and you'll need to prepare that too, not just blank screens. Developers are accustomed to thinking of data in terms of structure (the account description is a 20-character ASCII string) rather than content (Mike's IRA). But the content is what users care about. They enter data that is meaningful to them, and they expect the interface to reflect the information they entered. The interface is really just a means to an end, so users are likely to pay more attention to the data than they do to the widgets that make up the interface.

Often it's important for users to see how their data will appear elsewhere in the system. If a user is creating an online classified ad by filling out a form, he or she will probably want to preview that ad in the same format the system will use to display it. If you ask users to simply imagine something ("It'll let you preview your ad"), you could miss important feedback, such as, "Oh, wait—it took out those extra blank lines I wanted it to have."

The data you use should be realistic enough that users can interact with it in a meaningful way. This is especially important when users are domain experts in the subject matter of the interface. In the MathWorks cpselect case study described in Chapter 2, the team initially tried to use hand-drawn sketches to represent pictures of land masses in their prototype. They quickly learned, however, that this wasn't good enough for image processing tasks. The users got distracted trying to make sense of the sketches because image processing was what they did in their work. The team replaced the sketches in their paper prototype with aerial photographs, which were suitably realistic and worked much better. Similarly, if your users are accountants and you're testing a financial application, the numbers had better make sense.

Make sure the data "hangs together" in a consistent manner throughout the task. If you ask users to find all the three-bedroom houses for sale but you show them search results that include some four-bedroom ones, users might wonder what they did wrong. (Of course, if you're clever with the removable tape, perhaps you can cover up the four-bedroom houses, thus solving this problem on the fly.)

You should also include a realistic degree of complexity in your paper prototype. If your online hardware store sells 37 cordless drills but you pretend there are only 3, you'll design the drill page in a way that may not scale. This is not to say that you need the product pages for all 37—you might constrain the task in such a way that the users are likely to look at only 6 of them.

Where do you get realistic data? Some companies have suitably complete databases that are used in quality assurance testing, or perhaps Marketing has something they use for demos. But if your test databases contain mostly nonsense filler such as "test1" and "asdfasdf," you're better off creating your own. If you are having trouble coming up with a realistic scenario, talk to the people in your company who have contact with users. Just don't use a database that has real people's information in it—that's too realistic!

Divide and Conquer

The best way to create the prototype depends on the size and composition of the team and how far along the development is. At one extreme, if you're making a prototype of an existing interface and it has a straightforward sequence of screens, you could simply make screen captures and print them out as the basis for your prototype (this is a great task for an intern). In this case you might adjourn the meeting until the next day, when you'll have the screen printouts, and then do your first walkthrough.

If the interface doesn't exist yet or is being substantially changed, there's some design work required. So divide and conquer: Have each person on the team put their initials on the list next to the screens they'll prepare. Leave the list where everyone can see it so that they all know who's working on what—people may want to pass along ideas or collaborate.

This divide-and-conquer idea initially makes some people nervous because it feels like I'm advocating design by committee, a committee that may include some nondesigners. But in practice this is not what happens. The paper prototype is created by the core team, which by definition includes those responsible for the interface, so the prototype always remains in the appropriate hands. Most prototypes have some screens that are well defined and relatively easy to create, such as

the browser background or log-in screen. What I've found is that that team members with less design expertise tend to sign up for the easier pieces, leaving the heavy lifting for the lead designer(s). The end result is that the work is split up in an appropriate way and everyone benefits by becoming familiar with the interface. (And if a suboptimal design idea does manage to creep in, usability testing will weed it out.)

It isn't wrong to have the whole prototype created by just one or two people if that's what you're most comfortable with. But it might be advantageous to divide up the work as I've described here, or even to try parallel design.

Parallel Design

Sometimes you have to come up with many mediocre ideas to get a few good ones. Because paper prototypes aren't confined to a computer screen, they lend themselves to collaborative activities in which several team members spread out their sketched designs on a wall or table. As explained later, it can be helpful to *generate* several variations of a design as a means of finding good ideas, but before you *evaluate* (usability test) the design you should probably pick just one.

In a technique known as parallel design, product team members work alone or in small groups to generate different designs and then review everyone's ideas. Interestingly, parallel design seems to facilitate the identification and development of successful ideas, even when they're contained in not-so-good designs (McGrew, 2001). Thus, parallel design with paper prototypes can provide a way for non-designers (especially those who've had a lot of contact with users) to make meaningful contributions to the interface. In the Of Interest box on p. 149, Dr. Bob Bailey describes how he used parallel design in a course on user interface design.

Avoid Competition

Although it can be a fruitful exercise to create several variations of an interface, you need to know when to stop. One of the caveats of parallel design is the temptation to turn it into a "use-off."* It's rarely productive to foster competition within a

* This is my term for the usability equivalent of a cooking competition, where prototypes are pitted against each other in usability tests.

Of Interest . . . Parallel Design

Dr. Bob Bailey, Chief Scientist for Human Factors International (available at www.humanfactors.com)

Several years ago I taught several "hands-on" courses on user interface design. In one exercise, students were given a specification, and used a prototyping tool to create a simple system. After the design solutions were completed, each individual in the class used everyone else's proposed systems to complete a task. Having experienced everyone else's ideas, the students then made changes to their original prototypes. The revised interfaces were always better than the original. The three most interesting observations from these classes were

(a) how many unique ideas (creative design solutions) individual students had initially,

(b) no matter how good their original interfaces, every one could be improved, and

(c) how quickly students found and perpetuated good design ideas in their own products.

team because this can get people focused on winning rather than collaborating to produce the best design.

There's also another reason to avoid the use-off mentality. In essence, all usability problems have one root cause: The development team didn't know something that turned out to be important. This ignorance might be about what users needed, how they would interpret a term, how they would behave, or any other sort of missing information or incorrect assumption. Until you start doing usability testing, you don't know what you don't know. If you have several versions of an interface, it's likely that all of them will share some important flaws—based on all that stuff you don't know yet.

So although parallel design can be a useful technique in coming up with ideas, I recommend that the core team settle on one design before the first usability test. When two or more ideas seem equally good, start with the one that's easiest to implement. If it works, you're done. If not, you can try something more elaborate.

Existing versus New Design?

Sometimes a debate comes up about whether it's better to test a prototype of the existing design or to scrap it in favor of a new approach. Neither answer is wrong—

a paper prototype can evolve from any starting point—so here are some factors to consider:

- **The cost of mistakes is low.** If you try something new and it flops, you haven't wasted much time. And often you'll learn information that will help you even if you keep the current design. For example, one team prototyped a wizard to help with a configuration task. They weren't sure whether their development schedule would allow time to implement the wizard, but they still got feedback about terms, concepts, and what assistance users needed to make decisions along the way. All this information was useful even though, as it turned out, they had to stick with a variation of the current design.
- **Don't redesign something you haven't tested.** If you don't have consensus on what's wrong with the current version, you might want to use it for the first couple of tests until you understand the problems you need to solve and then do your redesign. It's easy to fall into the trap of reworking and polishing a design before users ever see it, only to have the first usability test reveal some huge issue that forces you to redesign it. On the other hand, if your customers have already told you what's wrong and you have ideas about how to fix it, go for it. (As a developer once said to me, "We know this part is bad—there's no point in getting 10 people in here just to tell us it's bad.")
- **Limit time, not creativity.** You don't want to spend long stretches of time designing without getting user feedback—that would defeat the purpose of paper prototyping. One way to manage this is to establish an upper limit for the amount of time you'll spend creating your prototype. When you've reached that limit, test what you've got.

One advantage of paper prototyping is that you can prototype something without regard to whether it can actually be built. This is also one of its drawbacks. The degree to which technical constraints should be considered depends on whether you're prototyping something you plan to release in 3 months or 3 years—the longer your time frame, the more you can focus on what users need and then figure out later if and how to build it. There's a similar argument for adherence to interface style guides. If your company has a corporate style guide and it's not negotiable, it's probably best to have your paper prototype conform to the style guide. But if you're trying to make a case for changing those guidelines, you might deliberately do something different to determine whether it will work better.

Hand-Drawn versus Screen Shots?

Perhaps there's an existing version of the interface and someone suggests that you start by making screen shots. Following are four factors to consider.

- **How much do you expect to change it?** You might as well use a screen shot for something that you know you can't change, such as a screen from the operating system. However, for screens that you expect to evolve (which is probably most of them), you might want to draw them by hand so that hand-drawn changes won't stick out the way they will if you use screen shots.
- **Which method is faster?** Although it seems like it would be faster to create a prototype from screen shots, in practice this is not always the case. Consider the time to walk to the printer, wait for your screens to come out (half of them are stuck in the queue behind someone who's printing 20 copies of the functional spec), get waylaid by someone wanting to discuss yesterday's meeting, and so on. Ask yourself whether you could sketch some of the screens faster, especially the straightforward ones.
- **Where is the data coming from?** If your interface shows information from a database (for example, a product catalog) and you happen to have a suitable database on hand, printing screen shots might be the quickest way to get realistic data. On the other hand, if you have to make up the information yourself, it might be just as easy to draw the scenes by hand.
- **Will the printouts be readable?** Printouts of screen shots are sometimes difficult to read. Without getting into the technical details, the colors you see on the screen often don't come out quite the same on paper (or vice versa). Ink is a different medium than light. As shown in Figure 7.1, the gray background that works well on the screen may look muddy on a printed screen shot. Depending on the resolution of the printer, icons and images sometimes lose their clarity. So if you print screen shots and find that they're difficult to read, the easiest thing to do may be to redraw them by hand. You already have a design, so this doesn't take long.

Although I'm not aware of any empirical evidence, I believe that a mixture of hand-drawn and printed screens still adequately conveys to users that the design is in flux and thus encourages their feedback. I've also found that prototypes that start out with a nice, neat set of screen shots usually evolve into a collection of

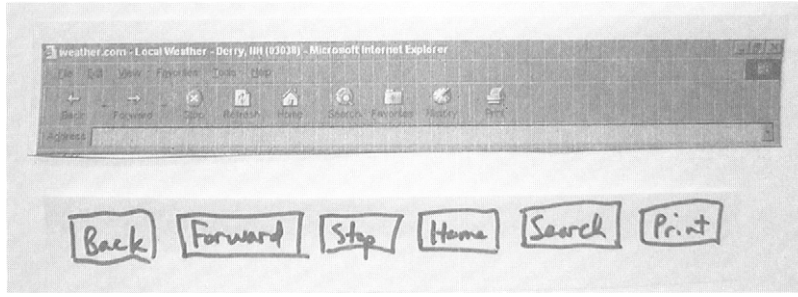

Figure 7.1 Two versions of browser buttons—one drawn by hand, and the other printed out. Notice that the hand-drawn buttons are easier to read.

screen shots, screen shots with handmade corrections, and some that have been redesigned and drawn by hand. So when screen shots are easy to make, I think it's okay to use them as a starting point and redraw screens if and when you find a need to.

If you make hand-drawn corrections to printed screen shots, occasionally users will be clever enough to realize that a hand-drawn correction is the thing they're supposed to focus on. (But there are still plenty of cases—I'd say the majority—where this doesn't seem to happen, probably because the screen still has several problems.) If you are concerned that a hand-drawn change might provide an artificial clue, either redraw several of the surrounding elements by hand also (this is where the wide removable tape is useful) or redo the screen.

Tips for Hand-Drawn Prototypes

Following are some tips to keep you from spending too much time preparing hand-drawn screens.

- **Neatness doesn't count (much).** Paper prototypes should be just neat enough to be legible, but no neater. It's fine to draw freehand instead of using a ruler. If someone else can read your writing, it's good enough. Resist the temptation to redraw something just to "neaten it up." Chances are, you'll want to change it as soon as you've tested it, so you might as well wait until you know what problems you're trying to solve. Don't refine a design you haven't tested because much of that effort will be wasted. See Figure 7.2 for an example.
- **Monochrome is okay.** As the saying goes, "You can put lipstick on a pig, but it'll still be a pig." Color can't save an inherently flawed design. In fact, it's a good

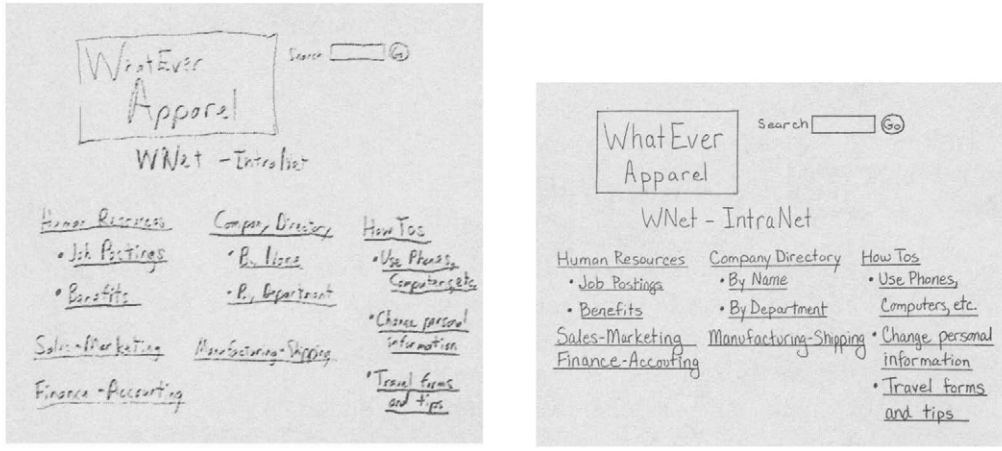

Figure 7.2 Although messy, the version on the left is legible enough to get user feedback. The extra time needed to create the neater version does not provide sufficient benefit to justify the effort.

strategy to design an interface in monochrome first. Color and other visual refinements can be added later once you're sure your design isn't a pig. But if you want to stick with black, that's fine too.

Even if you aren't adding color to the design until later, you can still have some fun with it now. If you're the type who writes with a fine-point mechanical pencil (like I did during my engineering days), getting a package of colorful art markers might help you to think more creatively. Choose a color or two that suits your personality—users don't seem to mind when one screen is drawn in blue marker and the next one is purple. But if you'd rather stick with black, that's fine too.

◈ **Size (usually) doesn't matter.** If users will be writing data on the prototype, you need to make the edit fields large enough to accommodate their writing. Drawing the prototype a bit larger than life also helps observers watch the action. Other than that, size usually doesn't matter too much. It's okay if the prototype is not drawn to scale; some screens can be relatively larger than others. As a rule, the concept of "making it all fit" should be preceded by a good understanding of what "it" consists of. If you're still determining functionality and content, you can probably postpone your concerns about screen real estate until a bit later in the process, provided that you keep your designs within reason. If you need to keep an eye on space constraints, you might want

to use graph paper or simply count characters and/or lines as a proxy for space.

Greeking and Simplification

Greeking means using nonsense words to represent text. In a paper prototype, you can use squiggly lines instead of nonsense words to serve the same purpose. Greeking is not needed in many prototypes, but it's worth knowing about. Following are three reasons why you might use greeking—the first reason is the most common, and the other two are more unusual.

1. **You haven't finished designing the interface.** Because paper prototyping lets you test an interface before it's completely designed, there may be portions of the interface that haven't been created yet. For example, you might know that your software application will have View and Help menus, but you haven't yet given any thought to their contents. So you could create greeked View and Help menus where all the choices are squiggly lines (see Figure 7.3). If the user tries to go to something you've greeked, you tell them, "The squiggly lines mean that you understand everything here but it's not what you're looking for." Of course, if users keep gravitating to something you happen to have greeked, you might want to consider putting in what they're actually looking for.
2. **To simplify parts of the interface that you have no control over.** A typical example is the File Open dialog box. If a task requires the user to open a file,

Figure 7.3 You can use greeking for parts of the interface that aren't directly related to your tasks.

you can create a simplified version that contains just the file name they need for the task (Figure 7.4). There's no real benefit in watching people use a more realistic version of this dialog box if you're not going to make any changes to it. Also, greeking avoids the need to have users navigate through a file structure that is artificial to them, which is less interesting for you to watch compared with other things they could be doing.

3. **To prevent the user from getting drawn into the content.** For example, when realtors were usability testing a real estate application, they kept wanting to read the descriptions of the houses that we'd made up. Since we were trying to test only part of the functionality—navigation, search engine, sorting—and not the format of the descriptions, having house descriptions was a distraction. So we greeked the descriptions, making it apparent to users that the information represented a house description but preventing them from focusing on it. Figure 7.5 shows an example from a florist Web site.

You've probably seen greeking that uses the words "Lorem ipsum dolor sit amet . . ." which is actually Latin, and faked Latin at that, so I have no clue why it's called greeking. The classic lorem ipsum wording was concocted so that its ascenders, descenders, and word lengths would approximate English text. Thus, it makes better filler material for screen layouts than "text text text" or "content here." Greeking is sometimes used on Web pages with the assumption that someone else is taking care of the content—as a search on "lorem ipsum" will reveal, sometimes this assumption is mistaken!

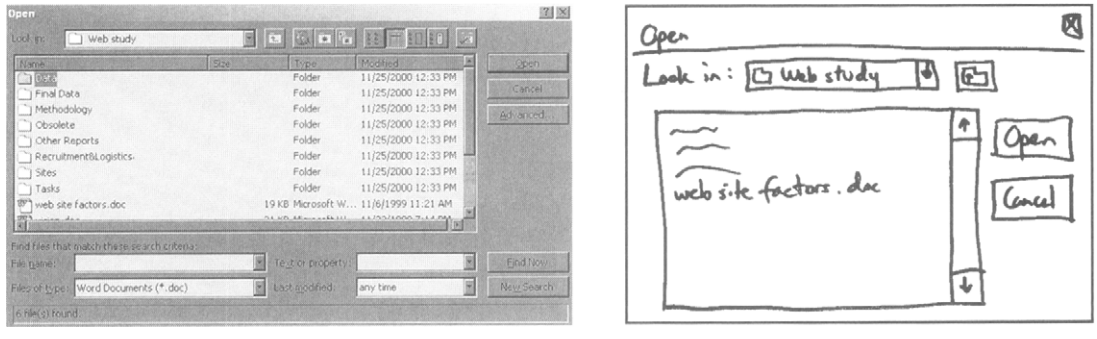

Figure 7.4 I usually use a simplified version of the File Open dialog box and greek the other files because we're not testing the operating system or the file structure.

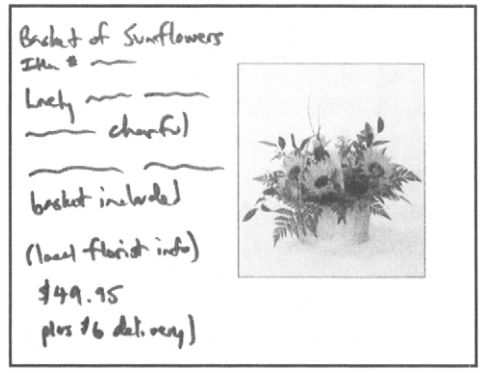

Figure 7.5 You can use greeking for content if you don't have it yet or don't want users to be distracted by detail that's irrelevant to what you're testing.

When people first learn about greeking, they're often tempted to overuse it. The previous examples should be considered as tactics to use in special cases rather than something you'd do as a matter of course. Don't greek parts of the interface that are relevant to what you're trying to test. For example, in a drop-down menu, you wouldn't want to greek all the choices except the one you want the user to select—that makes the task artificially easy, and you might fail to learn something important. It's also important in many cases to have realistic content in the prototype because it's interesting to watch users make decisions based on that content.

Using Screen Shots

Assuming you want to use some screen shots in your paper prototype, here are some things you might want to be aware of as you do your preparation:

- **Enlarge so that observers can see.** This may not be necessary if you have a really good camera setup or only a couple of observers, but if you have several people sitting around a table, most won't be able to read an actual-size paper prototype. It can also be difficult for users to write in edit fields that are sized for the screen, especially with a transparency marker. You might want to enlarge the screens by about 30% to 50% by using a copier.
- **Remove defaults** by whiting out radio buttons and checkboxes and then replacing them with versions on removable tape as shown earlier in this chapter. Radio buttons may not be too much of a problem because it's fairly clear that

the removable tape supersedes the original default state of the button. If the user unchecks a checkbox, however, it could introduce confusion later in the task if your prototype still shows an X in the box.

- Web-specific: **Clear link history** in the browser before making screen shots. If you don't, you're practically blazing a trail to show users the correct path—the visited links usually appear in a different color, even if you're printing in grayscale. One of my colleagues ruefully reported that her users quickly caught on to the visited links as being the correct path, and it undermined most of the value of the usability test.
- Web-specific: **Capture a whole page at once.** Not all screen grab utilities can do this, although this feature is increasingly common. SnagIt from TechSmith is an example of a utility that has an auto-scroll feature, or you can simply print from the browser (as explained earlier, you don't need to include the browser frame and controls if you create a browser background). Definitely avoid any tool that requires you to paste partial screens together manually.

When you take these factors into account, you may find that it's just as fast to sketch some screens by hand. Usually the best course of action is to do whatever is fastest to get your prototype together, then modify it as needed as a result of usability testing.

Separating Elements

Most interfaces, whether they're Web pages or software, have some parts that remain visible most of the time. For example, software applications typically have a row of menus and icons across the top, and perhaps some at the bottom. Some applications use a left-hand tree structure that expands and collapses. Web sites often have a persistent row of links at the top, a left navigation panel (that sometimes changes at lower levels of the site), and a content area.

Your first impulse might be to make screen shots of each whole screen, one for each permutation you expect the user to see. But if you decide to make a change to a persistent area of the screen, such as the left navigation panel, you need to make that same change on every page. Thus, it's often easier to cut up screens into their main components to facilitate revising only part of the screen, as shown in Figure 7.6. (Sometimes it can add a bit of confusion to have several pieces representing one screen, but simply telling the user, "This is all one screen" usually does the trick.)

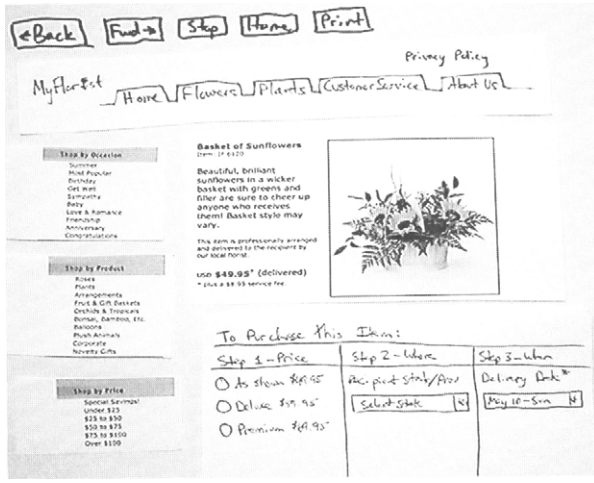

Figure 7.6 This Web page started out as a screen shot but was cut into pieces to facilitate site-wide changes. The revised tabs and ordering information (which were redrawn by hand in this example) can easily be used on every product page.

How Much to Prototype—Anticipating Paths and Errors

There are often several correct ways to complete a task, not to mention a seemingly infinite number of ways for things to go wrong. You should prepare enough of the prototype to accommodate things that users might reasonably do, but it's not necessary to anticipate everything that might happen in the usability tests.

For example, on a Web site, some users might use the search engine, whereas others navigate by clicking links. In this case it's reasonable to prepare both alternatives unless one of them simply isn't interesting for you to watch. But don't go overboard and try to prepare for every combination of terms the user might enter. Just pick the few search terms that you think are most likely and don't worry about those one-in-a-thousand edge cases. This may be inherently difficult for programmers to do because the exceptions and edge cases are often what make code complicated, and unanticipated edge cases are a common cause of bugs. But it's usually premature during paper prototyping to worry about making the code robust—you first need to make sure you have a design that works for users.

What about errors? It's pretty darn likely that some users will step off the path you envisioned for accomplishing the task. Prepare to handle the errors that seem likely, but don't make yourself crazy. It's always possible during a usability test to jot down an error message or even just tell the user what it would say. As explained

in Chapter 9, there are ways for the test facilitator to respond if a user tries to do something you haven't prepared for.

Avoid putting too much effort into your initial prototype. A good rule of thumb is that you should feel about 50% ready before your first internal walkthrough and 80% to 90% ready before your first usability test. If you feel 100% ready, chances are you've put too much effort into an interface that will need to be changed anyway.

Organizing the Prototype

No doubt about it, paper prototyping is messy. One of the challenges is organizing all the pieces of paper so that the Computer can quickly find the next one. There's no right way to organize a paper prototype—each Computer will develop his or her own system for locating pieces quickly. But here are some tips.

- **Ordering . . . not!** It's usually not possible to determine the exact order in which screens will be used. Even if there's an expected sequence of steps, users may back up, jump ahead, do things that cause error messages, look up something in help, and so on. But if you can be fairly sure that users will view a series of screens in a fixed order (for example, a tutorial), it may be helpful to number them on the back.
- **Table or three-ring binder.** Most Computers I've worked with will spread out piles of paper on the table, but some people use a three-ring binder to hold all the screens and flip back and forth to the appropriate page. I've never used a binder because it seems a bit limiting—you can't look at more than one thing at a time and it may take longer to find the next screen when you can't see everything. On the other hand, a binder is more portable, so do whichever works for you.
- **Organization by task versus function.** Sometimes it can be helpful to organize the pieces by task—everything for task 1 goes in a folder (or stack), everything for task 2 in another folder, and so on. However, some pieces may be needed for multiple tasks. If you don't expect the piece to change (say, the File Open dialog box), you can simply create multiple copies of it and put one in each folder. If it is likely to change, then you have to remember to create multiple copies of the revised version as well. This can be a pain if the interface is undergoing rapid iterations, so you may be better off creating only one version of each piece and putting it in the folder where it will first be used. In later

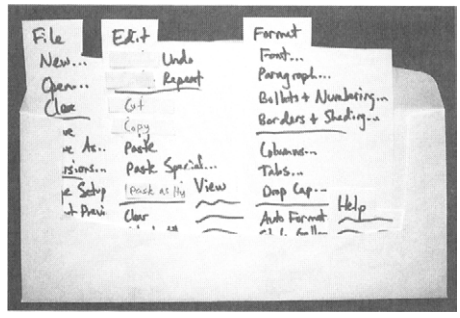

Figure 7.7 An envelope is a handy way to organize drop-down menus.

tasks, take pieces from the folders for earlier tasks as you need them. At the end of the test, take a couple minutes to put everything back where you'll want it before the start of the next test.

- **Use of envelopes.** Drop-down menus are usually written on small pieces of paper, so they can easily get lost in a folder full of larger pieces. One idea is to line them up side-by-side in an open envelope so that you can quickly find the one you need. If the pieces are small, you can even cut down the front of the envelope so that they're all visible, as shown in Figure 7.7.
- **A "Gallery of Annoying Pieces."** The smaller the pieces are, the more likely they'll get lost. This goes double for anything that's on removable tape—in the course of leaning across the table, the Computer invariably ends up with errant prototype parts stuck to his or her elbows (the record I've seen so far is three). To minimize this problem, designate a piece of paper as the "gallery of annoying pieces." All the small pieces—including those that come loose during the test—go on that piece of paper where they're easier to find later. See Figure 7.8.

Design Reviews

There can be considerable benefit to product teams in using a paper prototype to walk through the tasks themselves, without users. I'm going to distinguish between two types of walkthroughs—an *internal walkthrough*, which is used to prepare the prototype (and team) for usability testing, and a *design review*, where the

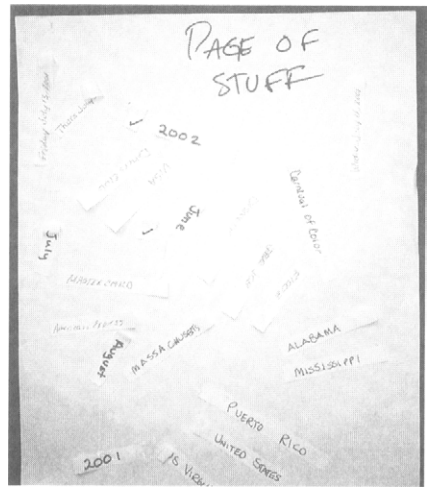

Figure 7.8 Give all those little sticky pieces a home so that they don't end up stuck to your elbows during a usability test. I call this the "gallery of annoying pieces," but this team has titled their version "Page of Stuff."

paper prototype is used as a means of discussing the interface with stakeholders. You may hear these terms used differently than I've defined them here, and to some extent they are overlapping concepts. This section briefly discusses design reviews; the rest of the chapter focuses on internal walkthroughs.

Many of my clients have told me that just the process of creating tasks and the paper prototype made them look at the interface differently—even before the first usability test, it's already been a useful exercise. Some product teams create paper prototypes and use them internally in design reviews, without ever conducting usability tests. A paper prototype can spur productive discussion, especially for those who may have difficulty visualizing the behavior of an interface from a specification. Design reviews using a paper prototype enable the product team to see the interface from a fresh perspective—they'll notice annoyances like cumbersome switching back and forth between two screens or a link that's missing. They may also identify larger issues, such as a missing requirement. Although design reviews are not a substitute for usability testing, they can help product teams identify and fix problems in the interface. So consider taking a paper prototype to your next interface review meeting.

From the Field: Paper Prototypes as a Design Aid

"One of our interfaces is used in an operating room to record data on cardiac implants. The path to get to one screen was too long given the time available to enter the data dictated by the doctor during the procedure. It's also challenging when the user has to enter data on multiple devices. Paper prototypes allow us to lay out a screen-by-screen process flow and identify where we may consolidate screens and tweak the order of screen progression."

Phillip Hash, Principal Human Factors Engineer, HiddenMind

Internal Walkthroughs

An internal walkthrough as I've defined it is an activity that helps the product team prepare to conduct usability tests with the paper prototype. A walkthrough is similar to a usability test in that you have a prototype and tasks, but in this case there are no outside users. Instead, one team member acts as an "expert user," performing the task the way the product team expects it to be done.

A walkthrough is a practical exercise that helps the team:

- Identify parts of the prototype that still need to be created.
- Prepare for different paths the user might reasonably take (correct or otherwise).
- See how the interface is intended to work, which is especially useful for those creating documentation or training or who are new to the development team.
- Give the Computer practice in manipulating all those pieces of paper.
- Identify issues pertaining to technical feasibility. Typically, these are researched after the walkthrough.

You should hold your first walkthrough as soon as you think you have most of what you need to get through at least one of the tasks. Depending on the size of the team working on the prototype, it may be easier to wait until you're ready to walk

through all the tasks at once, or you may want to tackle them one or two at a time. This also depends on the complexity of the interface and the confidence you have in the design. If you're creating something new, it's often useful to put the pieces together sooner rather than later, just to make sure everything hangs together.

How to Do a Walkthrough

To hold a walkthrough, you'll need a table that's large enough to spread out the prototype. Assign people to the following roles:

- **Computer.** The person who organizes and manipulates all those bits of paper (may be more than one person).
- **Expert user.** A product team member plays this role, performing the tasks as an expert (someone who understands the interface) would be expected to do. It's not part of this role to make bizarre mistakes or other radical departures from the norm—leave that for the real users!
- **Scribe.** This person writes down all the missing pieces, issues, and questions that come up. It's not always the best use of time for the entire team to discuss each issue on the spot and come to a decision. The scribe makes a list so that these things can be addressed after the walkthrough. On small teams, the scribe role may be combined with any of the other roles.
- **Facilitator.** Strictly speaking, a walkthrough doesn't need a facilitator, although it doesn't hurt to have a designated person lead the session and make sure it doesn't digress too much. Whoever plans to facilitate the usability tests should attend some walkthroughs in preparation; walkthroughs are a great way to become familiar with the functionality and behavior of the interface. For new facilitators, it's also an opportunity to practice.

Naturally, other members of the product team can also be present to help make missing prototype pieces, identify potential issues, learn about the interface, or simply to get a better understanding of paper prototyping.

As the expert user walks through the tasks, questions and issues will arise and the scribe should write these down along with missing prototype pieces. At the end, the scribe reads the list and team members decide who will do what before the next walkthrough. Agree on the time of the next walkthrough before dispersing to do individual work. In my consulting practice, I've found that 15 to 30 min-

utes is usually sufficient for people to research a couple of questions and make a few screens—in other words, to make enough progress that it's worth doing another walkthrough. If you allow too much time, you risk losing your momentum.

Caution: This Isn't a Usability Test!

There's often a strong temptation to turn a walkthrough into a premature usability test by asking a co-worker to play user. Typically, someone will say, "Hey, why don't we get so-and-so to be the user? She's never seen this before, so she'd be great." Resist! There are some problems with this, both obvious and subtle:

- **You aren't ready for prime time.** The reason for walkthroughs is to *prepare* for usability testing. You're certain to find unresolved issues, some of which you'll end up discussing on the spot, which is fine for those of you who are working on the interface, but a waste of time for the person you asked to help you. He or she has better things to do than listen to you debate whether a button should be disabled or how you'll handle a particular error.
- **Co-workers aren't real users.** Co-workers are usually not representative of real users, even if they talk to users every day or used to be users themselves. Your co-workers know stuff that real users don't, such as your company's business strategy, acronyms, brand, other products, and so on. Very likely, your co-workers know more about your interface—even if they haven't seen it—than real users do. Therefore, the behavior of a co-worker usually isn't a good proxy for what real users will do. (Possible exception: You're designing an intranet or other interface specifically intended for employees of your company.) And if you bring in a co-worker who truly "knows nothing" about what you're doing, then you may run afoul of the next issue.
- **There are ethical and legal considerations.** When you usability test with internal employees you actually have *more* ethical and legal responsibilities than if they were strangers. It's one thing if you have a bad test (one where the user feels foolish or appears ignorant) with a stranger, but it's quite another if you have to see that person in the cafeteria every day. It's even worse if that person is your boss! There are even potential liability issues: If a co-worker participates in a usability test and feels (correctly or otherwise) that their performance was perceived poorly by others, you could have a lawsuit on your hands. Of course, the likelihood of this is small, but you should think twice before asking someone outside the product team to play user.

At one company that shall remain nameless, a person coming in for a job interview was asked to participate in a usability test instead when the scheduled participant didn't show up. The person believed that the usability test was part of the job interview and became distressed about being placed in a situation that made him look bad. Lawyers got involved. The case was settled out of court, but it serves as a cautionary tale about the risks of conscripting people at random to participate in usability tests.

- **You don't want more opinions.** I've yet to work at a company where there's a shortage of opinions. The whole purpose of usability testing is to gather data from real users, not yet another internal opinion. It may not be wise to redesign an interface solely on the basis of internal feedback. If a co-worker gives you feedback that you don't want to act upon (at least not yet), then you're put in the awkward position of explaining why you're apparently ignoring the valuable advice you asked him or her to give.

Keeping the aforementioned cautions in mind, sometimes it can be valuable to ask a co-worker to play user for another purpose: to introduce them to the concepts of paper prototyping and usability testing. Some of my colleagues have reported very positive results from inviting an influential manager or VP to play user after first carefully explaining the technique of paper prototyping and the purpose of the walkthrough. There's more on the topic of convincing skeptics in Chapter 13.

Who Should Be the Computer

People sometimes think that you have to be a software developer to play Computer, but that's not necessarily the case. In fact, it can be valuable for a tech writer or trainer to be the Computer because this gives them experience with the interface. It's true that the Computer needs to understand how the interface behaves in response to the user's inputs, but one can usually pick up the necessary knowledge simply by watching someone else walk through the tasks a couple of times. Just as the interface isn't expected to be perfect, neither is the person playing Computer—if the Computer makes a mistake during a usability test (which is not uncommon for more complex interfaces), there's usually another team member present who will notice and help get things back on track. (It can also take some of the pressure off users to see that the Computer makes mistakes too.)

When practical, it's good to have more than one person who is prepared to be the Computer. You don't want to cancel a usability test because your Computer is home with the flu. Also, it's fairly demanding to be the Computer. In particular, it's hard for the Computer to also take notes, which is an important thing for the lead designers to do. So try to spread out the effort a bit. Some teams have two or three people who take turns playing this role, or they may have different Computers depending on the task. I've done paper prototype tests where Person A kept all the pieces for tasks 1 and 2, Person B was responsible for tasks 3 and 4, and so on. (While the Computers are changing places, I joke with the users that we're giving them a hardware upgrade!)

Some teams have a *Co-processor* work with the Computer. The Co-processor finds pieces, puts pieces back when they're no longer needed, writes things on the removable tape, and otherwise helps keep the prototype from becoming a disorganized mess. For example, when a user clicks "Add to Cart" on an e-commerce site, the Co-processor prepares a piece of removable tape to represent how that line item will appear in the shopping cart (see Figure 7.9).

When (and When Not) to Redesign

Sometimes even an internal walkthrough is enough to expose major problems in the interface. The mere act of performing the steps as users are expected to do can reveal aspects of the interface that are cumbersome or even technically impossible. When this happens, it's appropriate to come up with an improved version of the design even before the first test. It makes sense to redesign the prototype if the

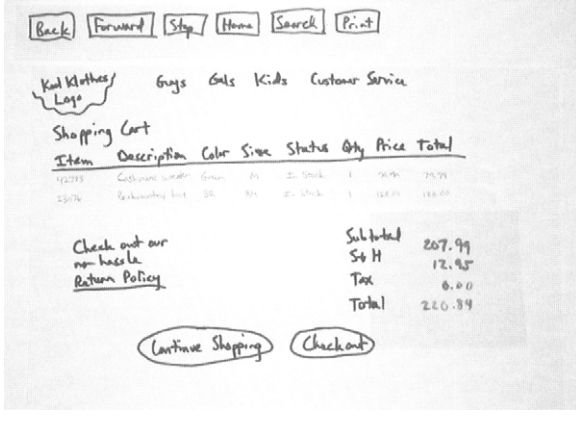

Figure 7.9 The Co-processor can help out by doing things such as writing the line items for the shopping cart based on what the user selects and recalculating the total.

team agrees that the issues found during walkthroughs are valid and serious. (As an example that may seem extreme but that occasionally happens, sometimes even one of the lead designers can't do a task.)

On the other hand, you should resist the temptation to redesign based solely on someone's opinion or the desire to try a different approach without understanding the strengths and weaknesses of the existing one. The problem with doing redesign before testing is that you don't yet know what you don't know. Once you watch users work with the prototype, you'll better understand the problems you're trying to solve. So if design ideas come up, jot them down and save them until you have some data from testing—then you'll know which of your ideas you truly need.

The Final Walkthrough–the Usability Test Rehearsal

The usability test rehearsal is basically another internal walkthrough, but with a few extra elements added. A rehearsal is not specific to paper prototyping—it's an important step in preparing for any usability test. Ideally, the rehearsal should take place the day before the first usability test. It often involves a larger group than those who worked on the prototype.

The purpose of the rehearsal is to make sure everyone (the Computer, facilitator, and observers) and the prototype are ready for usability testing. The facilitator runs the rehearsal in a similar manner to how he or she will conduct the usability test, but with several differences. Instead of focusing attention on the user (who in a rehearsal is still one of the product team members), the facilitator's goals during a rehearsal are as follows:

- **Familiarize observers with the prototype.** It's hard to get much use out of watching a usability test if you've never seen the interface before. That's especially true of paper prototypes because they change rapidly; the design you're testing tomorrow may not have existed yesterday. Because observers aren't allowed to talk during a usability test, encourage them to ask questions during the rehearsal. Once they understand the interface and the test procedure, they'll be prepared to get the most out of watching the usability tests. (Caveat: The observers aren't allowed to *redesign* the prototype at the rehearsal. If they have ideas, they should write them down and save them until the team is ready to make changes to the prototype.)
- **Collect people's questions.** Let everyone know at the start of the rehearsal that you want to gather all their questions about how well the interface will work

and what real users will do. For example, "Do users realize that they can sort by the column headings?" The test facilitator should write down these questions with the aim of getting as many answers as possible during the usability tests.

- **Estimate task timing.** Although it isn't possible to say with any confidence, "Task 1 will take 14 minutes," timing the tasks during the rehearsal will give you a sense of how long it takes to complete each task (assuming all goes well) and whether you have about the right number of tasks to fill the time allotted for the tests. During a rehearsal there is usually some discussion among the product team. To some extent, this discussion is a proxy for the amount of time that users will spend thinking aloud or getting stuck in the usability tests.
- **Decide when you'll intervene.** If there are doc/help writers on the team, they may be very interested to learn if the material they've written is useful. But many users don't go to the help or manual of their own accord. Discuss if and when the facilitator should intervene to suggest that they look something up. You might also want to agree on at what point the facilitator should step in and help users complete a task in the interest of moving on to the next one. (To an observer, it can be nerve-wracking to watch a user struggle with a task that is meticulously explained in help, to the point where that observer can no longer concentrate on what the user is doing. To forestall some of this stress, it can be helpful to discuss when and how the facilitator will assist users. You may also want to include this information in the Notes section of the task template.)

Note: Because I have 10 years of experience facilitating usability tests, I no longer need to have lengthy discussions about these things with the product team—they know that I am aware of what they're interested in, and they trust that I'll get as much of that information as I can. But if the facilitator and product team are unaccustomed to conducting usability tests, they haven't established that level of trust yet.

- **Create a "game plan."** It's not necessary to use the same tasks for each usability test—paper prototyping allows some room for improvisation in the test structure depending what you're learning. At the rehearsal, the facilitator should get a rough idea of which tasks are of greatest interest to the developers and under what conditions it may be appropriate to skip or reorder tasks. This is what I call a "game plan." An example of a game plan might be, "We'll do the first 3 tasks, and then depending on how much time we have, either go on to 4 or skip to 5." Like its sports counterpart, a usability test game plan is not a rigid structure but can be adapted depending on circumstances. For example, if a team member is very interested in task 5 but she can attend only two of the usability tests, in those sessions you might decide to do task 5 earlier to ensure she'll get to see it.

Pilot Tests

Unlike a usability test rehearsal, a pilot test is not something you'll do for every usability study, although there are situations in which one can be useful. A pilot test is a cross between an internal walkthrough and a usability test. Its main purpose is to refine the tasks and methodology used in the main tests (as opposed to a usability test, which is used to refine the interface). Unlike in a walkthrough, in a pilot test you bring in a representative user. Unlike in a usability test, you ask this user to give you feedback on the tasks and instructions, not just to do them. You can also time the pilot test to get a better idea of how long it will take users to accomplish the tasks. In a pilot test, it's appropriate to interrupt the user (politely, of course) to have a discussion with your teammates about how you want to change the tasks or methodology—this is not something you want to spend time doing in a usability test.

A pilot test is useful when you don't want to change the tasks during the usability study. For example, I tested a live Web site in the United States that was also being tested in Denmark. My Danish counterpart created the tasks, and I conducted the pilot test. I then reported back to her my results such as, "Task 2 will take more time than we thought, so we should allow up to 40 minutes. But task 3 and task 5 cover similar ground, so let's eliminate task 5." The pilot test helped us coordinate our methods so that our test findings would be comparable.

I reserve pilot tests for situations in which there is a high degree of rigor needed in the testing methodology, such as a competitive evaluation of two released products. Because a paper prototype is continually evolving, there is less need to hold the methodology constant, so I don't bother with pilot tests. However, there is no reason not to conduct a pilot test if you feel the need for one. In particular, a pilot test can be a good opportunity for a new facilitator to practice under the guidance of a more experienced one. Speaking of new facilitators, the next chapter explains the basics of usability test facilitation.