Prototyping for Tiny Fingers

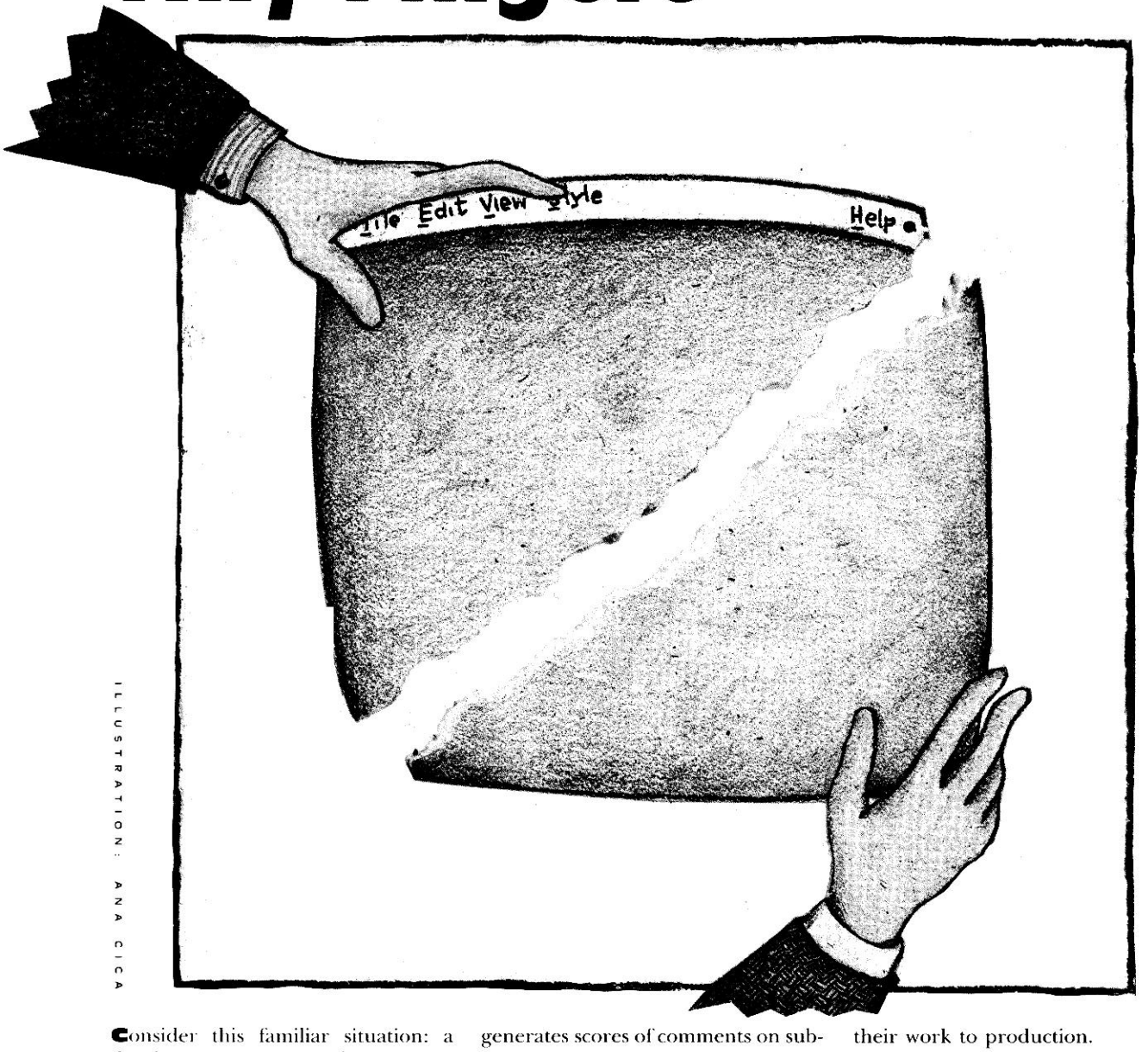


ILLUSTRATION: ANA CICA

Consider this familiar situation: a development team spends weeks designing an interface. They draw sketches on the board, discuss each point in detail, and finally specify a design. The design is either coded into the application language or simulated with a software prototyping tool. The result is finally shown to users for approval, in a session that generates scores of comments on subjects ranging from the basic metaphor to the choice of background color. The team just barely has time to incorporate these comments into a revised design before committing their work to production.

by Marc Rettig

Now consider a different situation, one I have witnessed first-hand over the past few months: a development team spends weeks designing an interface. During the first few days, they construct a paper prototype of their initial thinking about all aspects of the design, and test it with typical representatives of the user commu-

nity. One of them "plays computer," moving components of the paper interface around on the table in response to the users' actions. The others observe and take notes. After the tests they take a week to distill lessons from their observations, redesign the interface, and retest with several new users. This process continues until, at the end of the time allotted for interface design, the team has revised their design four times and tested it with many typical users.

This technique—building prototypes on paper and testing them with real users—is called low-fidelity prototyping or "lo-fi" for short. The value of prototyping is widely recognized, but as the first situation exemplifies, that value is not always gained in practice. If that has been your experience, you might want to try lo-fi prototyping, which requires little more in the way of implementation skills than the ones you learned in kindergarten.

The idea of lo-fi prototyping (a.k.a. "paper prototypes") has been around a long time. So long, in fact, that more than one person in the CHI community expressed surprise when I said I was planning to write a column on the subject. But I see this as a wonderfully simple and effective tool that has somehow failed to come into general use in the software community. I say this based on the success of the teams I've watched over the past several months together with the fact that this is the first commercial project where I've seen paper prototypes employed.

Paper prototyping is potentially a breakthrough idea for organizations that have never tried it, since it allows you to demonstrate the behavior of an interface very early in development, and test designs with real users. If quality is partially a function of the number of iterations and refinements a design undergoes before it hits the street, lo-fi prototyping is a technique that can dramatically increase quality. It is fast, it brings results early in development (when it is relatively cheap to make changes), and allows a team to try far more ideas than they could with high-fidelity prototypes. Lo-fi prototyping helps you apply Fudd's first law of creativity: "To get a good idea, get lots of ideas."

The Problems with Hi-Fi

For years developers have used everything from demo-builders to multimedia tools to high-level languages to build prototypes. Lo-fi proponents call these "hi-fi prototypes." They have their place: selling an idea, testing look-and-feel, detailed proof-of-concept, testing changes to an existing system, and so forth. I'm not suggesting we should stop building them. But they also have problems.

- *Hi-fi prototypes take too long to build and change.* Even with high-level tools, a fully functional prototype can take weeks to create. I have seen teams build a complete working lo-fi prototype in four hours. The goal is to get through as many iterations as you can during the design phase, because each iteration means improvement. If testing flushes out problems with the basic metaphor or control structure in a design, changing the prototype can again take weeks. This is what Debbie Hix and Rex Hartson, researchers and faculty members at Virginia Tech, call the "Software developer's dilemma." You can't evaluate an interaction design until after it is built, but after building, changes to the design are difficult.

Paper prototypes, on the other hand, are extremely fast to develop and the technique is very easy to learn. It is the fastest of the so-called rapid prototyping techniques. To make a broad generalization, interface designers spend 95% of their time thinking about the design and only 5% thinking about the mechanics of the tool. Software-based tools, no matter how well executed, reverse this ratio.

- *Reviewers and testers tend to comment on "fit and finish" issues.* You are trying to get feedback on the big things: the flow of the conversation, the general layout of the controls, the terminology, the expressiveness and power of the basic metaphor. With a slick software prototype, you are just as likely to hear criticisms about your choice of fonts, color combinations, and button sizes. On the back side of the same coin, developers easily become obsessed with the prettiness-power of a good tool, and spend their hours choosing colors instead of coming up with new ideas.

In contrast, the hand-made appearance of a paper or acetate prototype forces users to think about content rather than appearance.

- *Developers resist changes.* They are attached to their work because it was so hard to implement. Spend enough time crafting something and you are likely to fall in love with it. Knowing this, team members may feel reluctant to suggest that their colleague should make drastic changes to the lovely looking, weeks-in-the-making software prototype. They would be less hesitant to suggest redrawing a sketch that took an hour to create.
- *A prototype in software can set expectations that will be hard to change.* Prototyping tools let you do wonderful things in a (relatively) short time. You can make something that looks like a finished product, fooling testers and even management into thinking how far you are along. If it tests well, you may wind up spending time on "reverse damage control," handling questions about your sudden lack of progress.
- *A single bug in a hi-fi prototype can bring a test to a complete halt.* To test effectively, your prototype needs to be complete and robust enough for someone to try to do something useful with it. Even with the coolest of high-level tools, building a prototype is still essentially a programming exercise—and we all know how hard it can be to get all the bugs out of a program. On the other hand, I often see teams correcting "bugs" in a paper prototype while the test is in progress.

A Trojan Meme

The spread of lo-fi design through my current project started with a visit from Jared Spool (with User Interface Engineering in Andover, Mass.). He and his associate presented the basic ideas, then put us to work in four teams to design and build a prototype of an automated menu for a fast food restaurant. For three hours we discussed, designed, sketched and glued, then ran the results in a face-off competition with "real users" and a "real task." That is, we brought people in from elsewhere in the building and told them, "you have $4.92. Order as much food as you can." The designs were measured by how quickly and efficiently people

Lo-fi prototyping works because it effectively educates developers to have a concern for usability and formative evaluation, and because it maximizes the number of times you get to refine your design before you must commit to code.

could use the interfaces without coaching from the designers. Between tests, each team had a few minutes to refine their interface.

We were all impressed with the results of the exercise. In about six hours we had learned the technique, designed an interface and built a model of it, conducted tests, and measurably improved the original design. That was four months ago, and now we have scores of people working on lo-fi designs, refining them through repeated tests with actual users. Interface sketches are lying all over the place, scans are put on the network for peer review, and terms like "affordance" and "mental model" are common parlance.

I call this a "Trojan meme" instead of just a "selfish meme" because it did more than reproduce itself through the department. (A meme is an idea—the mental equivalent of a gene, and selfish ones try to replicate themselves in as many minds as possible.) As it spread, it served as a vehicle for spreading a general appreciation of the value of usability design: developers saw first-hand the difference in people's reactions to successive refinements in their designs. Within days of designing an interface, they saw exactly how their work was perceived by people just like those who will eventually be using their product. The value of two important laws of interaction design was memorably demonstrated: "Know Your User," and "You Aren't Your User."

Testing for iterative refinement is known in the interface design community as "formative evaluation," meaning you are evaluating your design while it is still in its formative stages. Testing is used as a kind of natural selection for ideas, helping your design evolve toward a form that will survive in the wilds of the user community. This is in contrast to "summary evaluation," which is done once after the product is complete. With summary evaluation you find out how well you did, but you find out too late to make substantial changes.

Lo-fi prototyping works because it effectively educates developers to have a concern for usability and formative evaluation, and because it maximizes the number of times you get to refine your design before you must commit to code. To make the most of these advantages, the prototyping effort needs to be carefully planned and followed by adequate testing and evaluation. (It also helps to have someone who can enthusiastically champion the idea.) Hix and Hartson have an excellent chapter on formative evaluation in their book, *Developing User Interfaces*. If you plan to adopt any of these techniques, I recommend you read their book.

The rest of this is drawn from our experience over dozens of designs and scores of tests, notes from Jared Spool's workshop, and Hix and Hartson's book.

Building a Lo-Fi Prototype

1. ***Assemble a kit.*** In this decadent age of too many computers and too few paint brushes, it might be hard to get all the materials you need by rummaging through the supply closet in the copy room. Make a trip to the office supply store, or better yet, the art supply store, and buy enough school supplies to excite the creative impulses of your team. Here's a shopping list:

- White, unlined, heavy paper that is bigger than letter size (11 by 17 inches is nice), and heavy enough to endure the rigors of repeated testing and revision.
- Hundreds of 5-by-8-inch cards. These come in handy as construction material, and later you'll use them by the score for note taking during tests.
- Various adhesives. Tape: clear, colored, double-backed, pin striping tape, whatever. Glue sticks, and most importantly, Post-It glue—a stick of the kind of glue that's on the back of those sticky yellow notes. Rolls of white correction tape are great for button labels and hurriedly written field contents.
- Various markers—colored pens and pencils, highlighters, fine and thick markers, pastels.
- Lots of sticky note pads of various sizes and colors.
- Acetate sheets—the kind you use to make overhead presentations. Hix and Hartson swear by these as the primary construction material for lo-fi interfaces.
- See what you find in the architecture section. They have sheets of rub-on texture, for example, which could give you an instant shading pattern.
- Scissors, X-acto knives, straightedges, Band-Aids.

Just like kindergartners, lo-fi designers sometimes find inspiration in the materials at hand. So go ahead—buy that package of colored construction paper. The worst that can happen is you won't use it. Eventually your team will develop their own construction methods, and settle on a list of essentials for their lo-fi construction kit.

2. ***Set a deadline.*** There is a terrific temptation to think long and hard about each aspect of the interface before you commit anything to paper. How should you arrange the menus? What should be in a dialog box, what should be in menus, and what should be in a tool palette? When you are faced with a blank sheet of paper, these kinds of decisions crowd your thoughts all at once. "Wait," you think, "we haven't thought about this enough!"

That's exactly the point: no matter how hard you think about it, you

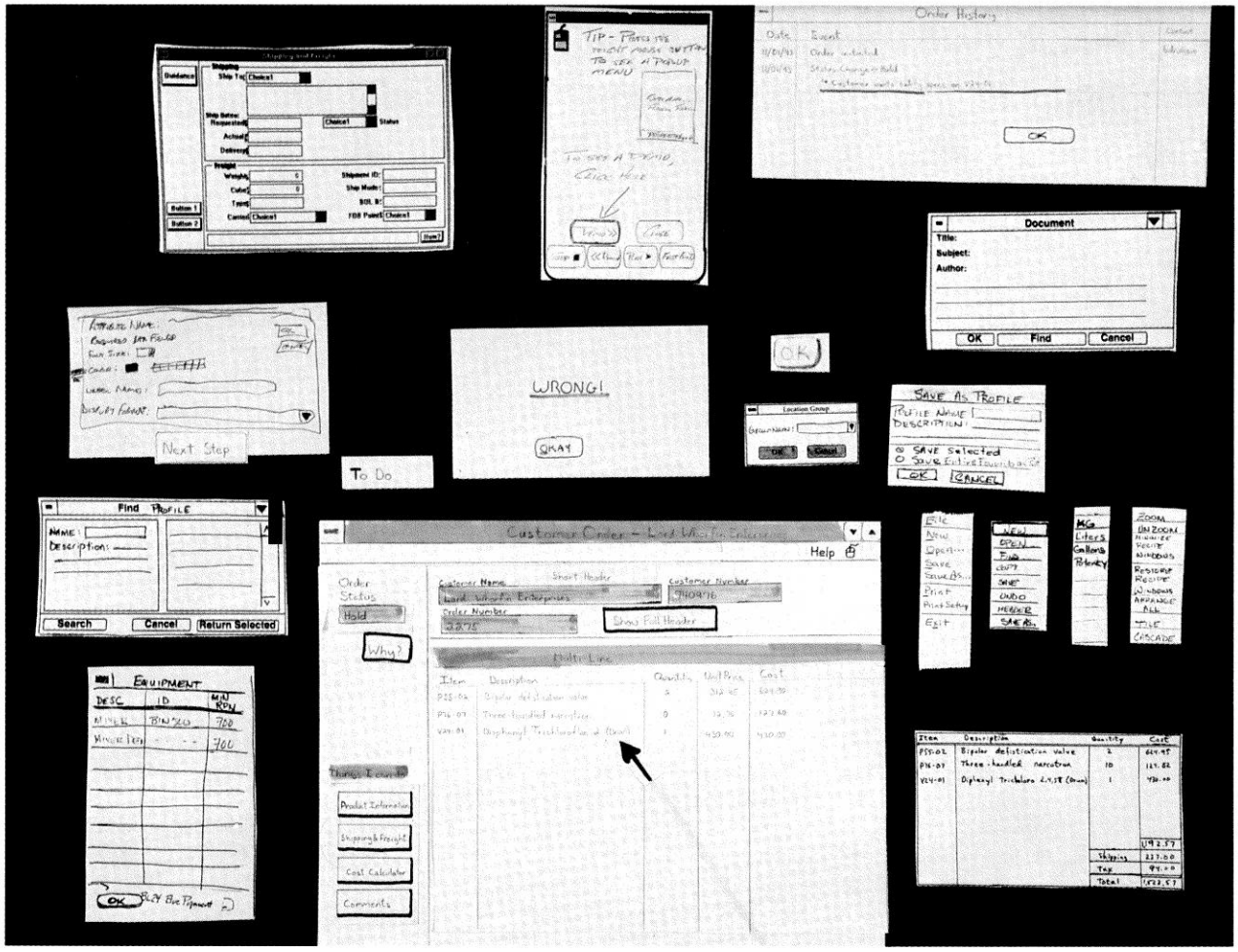


Figure 1. A few components of a paper prototype. The main window is in the middle, showing a few pieces of data added with strips of correction tape, and controls stuck on with Post-It paper. The window is surrounded by pop-up menus, dialog boxes, and sundry interface widgets.

aren't going to start getting it right until you put something in front of actual users and start refining your idea based on their experience with your design. So set a deadline. Obligate yourself so you are forced to take a preliminary crack at every important aspect of your problem. If you know four users are coming tomorrow expecting to work with your prototype, you'll have far less trouble with fixations on minor details. You will make some decision, any decision, about the most important aspects of your design, and get them down on paper for testing. Quality comes through iterative refinement. Get the big things right during lo-fi, and the little things will follow in future iterations.

3. ***Construct models, not illustrations.*** As you build, make something that will be easy to control for the person playing computer. Draw a genetic frame (a window frame, if you are building a windowing interface) on a big piece of heavy stock. Then make each region of the screen on a different piece of card stock. Anything that moves, changes appearance, or comes and goes, should probably be on its own piece of stock. Remember you'll eventually be playing computer, and you'll need to have a ready response for almost anything the user might do while trying to complete the test tasks. Make what you need to simulate menus dropping down, dialogs popping up, selection highlights, and so forth (see Figure 1).

Making a working model of an interface challenges you to be creative. Need to represent a long scrolling list? How about cutting slots in the dialog box and running adding machine paper through it? Want a working tabbed notebook metaphor? Cut up cards or file folders to make them the right size, draw the interface components below the tabs, and stack up the results so the tabs stagger the way you want. Use colored acetate to represent selection highlighting. Tape clear acetate handles to things that move around a lot, so you can handle them easily without obstructing the user's view.

The photocopier can be a lo-fi architect's best friend. Make yourself lots of blank windows, dialogs, buttons and fields, in several sizes. Get a fresh start by painting white-out over mistakes, then using the copier to produce a new unsoiled version.

You'll find that different people excel at different aspects of lo-fi construction: design, inventing ingenious paper implementations of wid-

gets, producing large amounts of data, or rendering artistic and attractive designs. Exploit these talents and divide the labor accordingly.

Construct a first version completely by hand. Sketch the widgets, hand-letter the labels. Don't even worry about using a straightedge at first. Just get the ideas down on paper. Test small details on one another, or drag people in from the hall for quick tests of alternative solutions.

Of course, hand-drawn sketches, no matter how carefully done, may not be appropriate for some testing situations. For example, a customer may be willing to let you test your design with actual users. They may understand the transience of the prototype, but you still want to make a good impression. You want to look sharp.

Some of the teams on my project have made remarkably attractive paper interfaces using components created with drawing software, then printed on a laser printer. Some of them build up a familiar look with elements taken from screen captures. To facilitate this kind of thing, they set up a library of lo-fi widget images: blank buttons of all sizes, window and dialog frames, scroll bars, entry fields, and so on. People print these out, resize them on the photocopier, and make them part of their standard lo-fi kit. Or they resize them on the computer, add labels, and print out a custom part for their work in progress.

This is an example of the kind of preparation that will help lo-fi prototyping become a normal part of your design process. Preparing a widget library, writing down guidelines, and taking time to train people will make everyone more enthusiastic and productive.

Preparing for a Test

However much care you take in building your prototype, the tests will be ineffective unless you prepare well for them. Be sure to attend to the following matters.

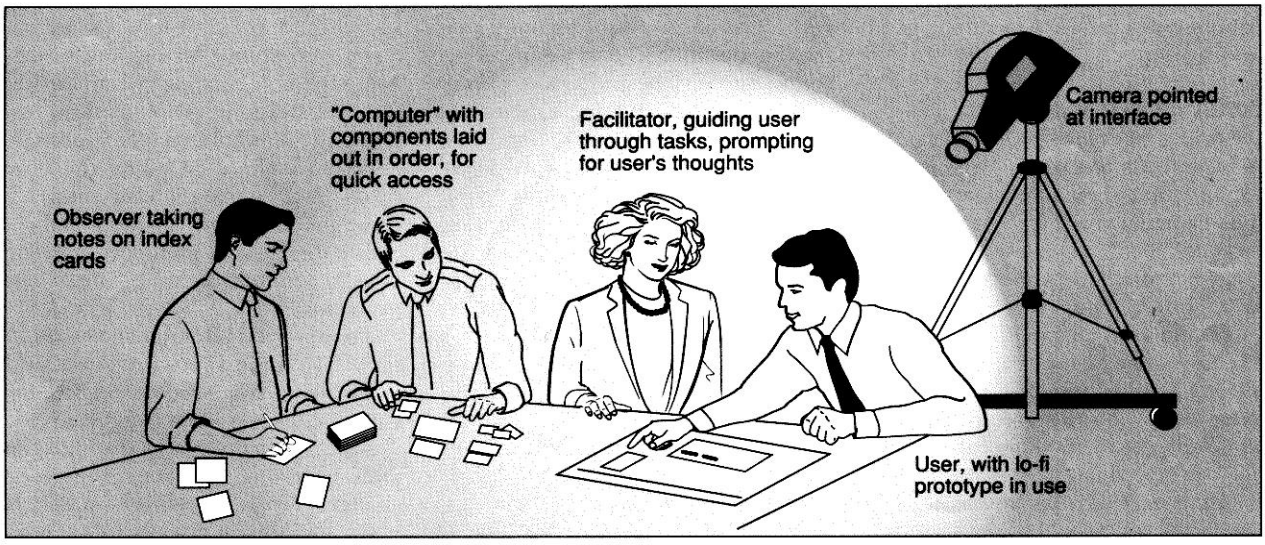
1. ***Select your users.*** Before you start designing, you should do enough user and task analysis to understand the people who will be using your software—their educational and training background, knowledge of computers, their familiarity with the domain, typical tasks involved in their job, and so on. Based on this study, you can look for pools of potential testers for your prototype. With a good user profile on hand, you can develop a questionnaire that will help to choose the best representative users from available candidates.

It would seem reasonable to arrange it so the people testing your prototype are the same people who will be using the final product. But bona fide members of the user community may be hard to corral for the time it takes to run a test, and using them may not be the best idea in the long run. Be sensitive to the political climate. People may feel threatened by the intrusion of a new system into their work (perhaps justifiably!), or there may be a competitive situation that makes your employer reluctant to expose new ideas outside the walls of your building.

Since you are looking for appropriate knowledge and skills, not job titles, you can often get by with "surrogate users"—people who fit the same profile as your actual clients, but free from whatever association that prevents you from testing with the clients themselves. I've heard of all kinds of tricks for attracting people to the test. Spool says he's done everything from running ads in the newspaper to recruiting university students to contacting local user groups. Anything to avoid using actual customers, employees, or friends and family. (The latter may be accessible, but there are a lot of things about sharing ties in the same social web that can conspire to damage a usability test. For example, testers who know you or the project may skew the results by trying hard to please you or do what they think you expect them to do.)

Finally, remember that no two people are the same, and your product's users may be a diverse group. Try to recruit testers that represent the whole range of characteristics in your target audience. Our practice has been to conduct at least one

Figure 2. A lo-fi testing session



round of testing in our office with surrogates, then go to the field for testing with the most typical end users we can find.

2. ***Prepare test scenarios.*** Write a set of scenarios, preferably drawn from task analysis, describing the product during use in a typical work situation. Design your prototype to support a few of these scenarios, narrowing the scope of your efforts to a reasonably small set of functions, but broad enough to allow meaningful tests.

If possible, ask someone to review the scenarios and sample data and tell you whether they look realistic. In our experience, people find a lo-fi interface more engaging—more realistic—if it shows data that looks familiar and we ask them to perform realistic tasks. This helps draw them into the "let's pretend you're really using a computer at your job" world, which leads to better tests. On the other hand, unrealistic scenarios and data can severely damage the credibility of your design.

3. ***Practice.*** Just as a bug in a software prototype can ruin a test session, so can a bug in a lo-fi prototype. That bug could be a missing component, a misunderstanding on the part of the person playing "computer," or even excessive hesitation and confusion because the team is unfamiliar with how to conduct a test. So to avoid embarrassment, conduct several dry runs before you test with people from outside your team. Each team member should be comfortable with his or her role, and you need to make sure you have the supplies and equipment needed to gather good information.

Conducting a Test

We find it takes four people to get the most out of a test session (see Figure 2), and that their activities fall into four essential roles:

- *Greeter.* Much the same as the usher in a church, the greeter welcomes users and tries to put them at ease. We have some forms we ask people to fill out—an experience profile, for example—a job the greeter handles while other team members are setting up for the test.
- *Facilitator.* Once the test is set up, the facilitator takes the lead, and is the only team member who is allowed to speak freely during the test. Facilitating means three things: giving the user instructions, encouraging the user to express his or her thoughts during the test, and making sure everything gets done on time. This is a difficult enough job that the facilitator should not be expected to take notes during a session.
- *Computer.* One team member acts as the "computer." He or she knows the application logic thoroughly, and sustains the illusion that the paper prototype behaves similar to a real computer (with an unusually slow response time). A pointing finger serves as a cursor, and expressions like, "I type 'half-silvered bicuspidors' in that field" substitute for keyboard entry. If the user touches a control, the computer arranges the prototype to simulate the response, taking care not to explain anything other than the behavior of the interface.
- *Observers.* The rest of the team members quietly take notes on 5-by-8-inch index cards, writing one observation per card. If they think of a recommended solution, they write it on the same card that records the problem.

Since all of these roles can be exhausting, we rotate them among the team when we conduct more than one session a day (and we very often schedule four sessions in a day).

Typical test sessions usually last a little over an hour, and go through three phases: getting ready, conducting the test, and debriefing. We begin with greetings, introductions, refreshments and general ice-breaking, trying our very best to assure people that the test is confidential, the results will remain anonymous, and their supervisor won't hear a word about whether or not they "got it." People often say things like, "Am I flunking the test? Am I getting it right?" To which we answer, "Don't worry, the question is whether or not we are flunking. The interface is on trial, not you. If you fail to understand something or can't complete one of the tasks, that's a sign of trouble with the design, not a lack of intelligence on your part."

While this is going on, someone positions a video camera (we tape all the sessions) so it points down over the user's shoulder to look at the interface and the hands moving over it. No one's face ever appears on tape.

During the test, the facilitator hands written tasks to the user one at a time. These must be very clear and detailed. As the person works on each task, the facilitator tries to elicit the user's thoughts without influencing his or her choices. "What are you thinking right now?" "What questions are on your mind?" "Are you confused about what you're seeing?"

While this is going on, the rest of the team members observe and take notes, and may occasionally interject a question. But they must never laugh, gape, say "a-ha," nudge one another, or otherwise display their reaction to what's happening to their careful design. This kind of thing can intimidate or humiliate users, ruining the relationship and spoiling the test. It can be terribly difficult to keep still while the user spends 10 minutes using all the wrong controls for all the wrong reasons. You will feel a compelling urge to explain the design to your users. Don't give in.

When the hour is over, we spend a 10-minute debriefing session asking questions, gathering impressions, and expressing our thanks.

Evaluating Results

Lo-fi or hi-fi, prototyping is worthless unless information is gathered and the product is refined based on your findings. As I wrote earlier, Hix and Hartson nicely cover the details of gathering and analyzing test data. We spend quite a bit of time (at least a day per iteration) sorting and prioritizing the note cards we wrote during the test sessions. Our method involves arranging the paper prototype on a big table, then piling the note cards next to its relevant interface component. Then team members divide the labor of going through the piles to summarize and prioritize the problems.

These sorted piles inform a written report on findings from the test, and form the agenda of a meeting to discuss recommended changes to the design. The team works through the piles and agrees on suggested changes, which are written on Post-It notes and affixed directly to the rele-

vant part of the paper prototype. Constructing the revised prototype becomes a process of taking each component, and following the recommendations that were stuck to it.

Try It

Hix, who for years has been teaching courses and workshops in interface design, says that people consistently enter the first lo-fi exercise with skepticism. After trying it they invariably say something to the extent of, "I can't believe how much we learned from this!" If this column is the first place you have heard about the lo-fi technique, one danger is that you will set aside this magazine with just enough skepticism that, however much interest I've managed to create, you will fail to actually try it.

Having seen other skeptics converted, I'm confident in recommending this technique. If you already have a working high-fidelity prototype, it probably isn't worth abandoning that course to switch to lo-fi. But if you are in the very early stages of design and exploring broad questions, or if you need to learn more now, lo-fi prototyping is just the tool to pick up.

Resources

Brown, D. *STUDIO: Structured User-Interface Design for Interaction Optimisation*. New York, Prentice Hall, 1994.

Hix D. and Hartson, R. *Developing User Interfaces: Ensuring Usability Through Product and Process*. New York: Wiley & Sons, 1993.

Nielsen, J. Finding usability problems through heuristic evaluation. In *Proceedings of SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, 1992, pp. 373–380.

Rudd, J. and Isenee, S. Twenty-two tips for a happier, healthier prototype. *Interac. 1*, 1 (Jan. 1994), 35–40.

Marc Rettig *is a senior architect at Andersen Consulting in Chicago. He can be reached at 76703.1037@compuserve.com, or 100 S. Wacker, Chicago, IL, 60606.*