

Usability Testing

In Which Doc Reveals the Benefits—and the True Cost—of Usability Testing

By Pete Bickford

It happens all the time. I'll be sitting through some demo when the programmer points to an object on screen and mouths those dreaded words: "This next part is kind of different, but it seemed like a neat idea, so we went with it."

The programmer then smiles gleefully and proceeds to show off his "innovation." In the past, these "kind of different" features have ranged from ghastly orange custom scroll bars to screens packed with every Hayes modem setting known to man. (Here, the programmer pointed to the hundreds of checkboxes and radio buttons while beaming, "See! You can set them all from one window!")

Now, Ma Bickford always said that if you can't say something nice, don't say anything at all (a restraint that is virtually absent from the human-interface profession). Nevertheless, I do try to be tactful, and usually respond with a hearty "Gosh! That's really . . . *interesting*. What did your usability testing say about it?"

At which point the programmer tends to utter a complex stream of syllables that goes something like, "Uh . . . err . . . well . . . you know . . . we . . . uh . . . well . . ." ending in ". . . didn't actually get a chance to do much testing on it."

I'm sure you, gentle reader, can imagine how surprised I am to hear this.

Testing Code Instead of Software

By now we've figured out as an industry that software ought to—oh . . . actually work if it's going to be sold to a customer. To ensure this, many shops hire software testers at a ratio of *at least* 1:1 to their programmers. The idea is that programmers aren't very good at testing their own code.

Strangely, the same shops often treat the actual *usability* of the software as a sort of luxury that can safely be squeezed into the workload of the programmers (whose lives tend to center around Toolbox traps and obscure data structures). And although most developers have by now at least heard the term *usability testing*, surprisingly little of their code will ever be exposed to it. In this way, it's much like the word *documentation*—some ten or fifteen years ago.

Lab Coat Not Required

If we are even a little kind, we can safely assume that programmers really do care that the software they create is actually usable. When asked why they don't *check* to make sure, the answers you'll hear are generally something like these:

- "Usability testing is too specialized/difficult."
- "We don't have the time."
- "Usability testing costs a lot of money."

The truth is that some of the most effective usability testing is incredibly simple, takes only a few minutes, and costs only about \$7 per subject.

All of these, I assure you, are vicious lies spread by the makers of video cameras and lab coats in conjunction with the ever-powerful one-way mirror manufacturers' lobby. The truth is that some of the most effective usability testing is incredibly simple, takes only a few minutes, and costs only about \$7 per subject. Moreover, there is documented proof that what you learn from doing usability testing pays itself back many times over by slashing your user support costs and helping you avoid design pitfalls.

A Brief Lesson in Conducting a Usability Test

The following are some instructions for carrying out a basic usability test:

Step 1: Find a user. A real user (as in, "someone who will be using your system"). If you can't find one of these, find someone a lot like the people who will be using your system—someone who has the same kind of technical and occupational expertise.

Caution: If you're developing an in-house system, resist the temptation to use a manager as a test subject, unless the actual system will be used exclusively by managers. Too many designers give in to this temptation, since the real users are considered "too busy" or "not important enough" to participate in a usability test. So instead they send their managers, a system is designed around *them*, and the result is something unusable by the rank and file.

Step 2: Set the user down with a drawing or prototype of your system. The important thing about prototypes is to remember that they are not "real." They should be designed quickly and thrown away quickly. The prototype only needs to be good enough to get the basic ideas across.

Step 3: Explain to users that you're there to find the areas of the system that are confusing or difficult, and that any place they run into trouble is an opportunity for you to make the system better. "Mistakes" are not the user's fault—they just point out trouble spots in the system.

Step 4: Ask users to perform a set task using the system. Explain to them that you will not be offering any help, and that they should “think aloud” so you can tell what they’re thinking as they try to work the system.

Step 5: Watch users quietly and note the areas where they do unexpectedly well or where they run into trouble. And here’s the hardest part: You absolutely must resist the temptation to give “hints” or point out parts of the system that they “overlooked.”

Step 6: At the end, ask them about the areas of the system that you noted, then thank them and give them a project T-shirt for their trouble.

Step 7: Use the results.

Total time for the test: usually under 30 minutes.

Total cash outlay: \$7 per user (for the T-shirts).

If you’re keen on spending money, you can hire consultants and do a lot of videotaping behind one-way glass; however, you’ll get 90 percent of the benefit by just using the above techniques. The important thing is to get out there and “just do it.” If you try an interface out on four people, you’ll usually find three of them hitting the same problems. Any such problem will need to be fixed (or, if there’s nothing that can be done, at least documented).

Win Friends and Change Minds

Usability testing finds problems with your product that you never could have guessed were there. For instance, you’ll sometimes discover that wording that is natural to you gives entirely the wrong impression to a regular user. Other times, you’ll find that buttons and menus that seem obvious to you are ignored or overlooked by users who either don’t notice them or don’t interpret them as being important to solving the problem at hand. It’s these little things that add up to a sense of anxiety and confusion on the part of the user. A few minutes spent fixing these things can make all the difference—but first you have to know that the problems are there.

Usability testing is also useful from a political level within an organization. For one, it’s the evidence that matters when trying to settle an argument between different design approaches. Engineers of good conscience can have hellacious battles arguing which interface design is the best for solving a given problem. Although designs can take hundreds of hours to code, a few hours of usability testing can often settle the question.

Another thing to remember is that usability testing can be *wonderful* public relations for your project. By taking the time to go out and recruit target users to help in your usability testing and product design, you’ll have gone a long way toward spreading goodwill and the sense that you really care about your customers’ needs.

The All-Important Paradox of Usability Testing

There is, however, one great paradox to usability testing: Although you can use it to find out from users what went wrong, it’s usually unproductive to then ask them how to make the design right. Users, as much as we love them, are not designers—they seldom have knowledge of the technical possibilities for solving a problem in the optimal way.

So, given a really unusable system, they’ll generally suggest a system exactly like it as a solution—with the one or two things that bugged them the most changed in some way. In the pre-Macintosh days, you often heard users ask for smaller type so that they could see more information on the screen at one time (remember 132-column screens?). Thankfully, the designers of the Macintosh looked beyond the immediate request (smaller characters) to the underlying need (to see more data) and created a system where you could view information using multiple windows. The lesson: While you need to notice what the user’s problems are, you as a designer are responsible for looking beyond the surface to discover the underlying issues and offering a real solution.

Keeping Us Honest

Well-organized development teams include marketers, engineers, graphic artists, documentation folks—and, yes, human-interface designers. Over the course of the project, it’s almost guaranteed that arguments are going to arise over some aspect of the human interface. And, though the human-interface designer will often have the best solution (I naturally offer myself as a shining example of this), nobody on the team has a monopoly on the truth.

In the end, there is only one judge of how good the human interface is, and that person is the user. By doing usability testing on your product in the development stage, you stand a much better chance of passing muster when your product is “usability tested” in the marketplace.

*Best wishes,
Doc*

This month’s article is a reprint of Pete Bickford’s June 1994 column. Unfortunately for us, Pete has left Apple to pursue other opportunities. We thank him for over four years of excellent columns on human-interface issues. Look for a book from him, tentatively titled Easy to Use: A Developer’s Guide to the Art of Human Interface Design, to be published later this year by Academic Press Professional. You can reach Pete at pbickford@human-computing.com.

Peter Bickford was a human interface senior scientist in Apple’s Developer Consulting Group. This article was originally published in AppleDirections, Apple’s monthly developer newsletter (<http://devworld.apple.com/mkt/adtop.shtml>). Copyright 1997, Apple Computer, Inc. Used with permission on the Bluecoat Reports Web site.