CHAPTER 14

User Testing on the Cheap

COLUMN 13: MARCH, 1990

One of the great myths of our time is that user testing is so time-consuming, expensive, and divisive that it should be primarily discussed, rather than actually undertaken. The truth is that user testing can be really, really cheap and really, really easy to do. It can even save you money—lots of it. And you don't need one-way mirrors and video cameras and all that other stuff that only keeps you from even trying. All you need is a couple of random people, a clipboard, and a computer.

User testing strikes fear into developers' hearts around the world. The conventional wisdom seems to be that such testing is (1) positively crippling and (2) properly reserved for the next release. The mighty "Communications of the ACM" once even published a paper claiming that the "costs required to add human factors elements to the development of software" amount to $128,330. (Mantei and Teory 1988)

Fortunately for the rest of us, Mantei and Teory are wrong.[1] You *can* spend that much, if you've got the budget and the time on your hands, but the major benefits of user testing can be had for a fraction of that cost. And this is not just my opinion. No, there are genuine scientists that agree with me! Before revealing their names, however, let us examine what might happen if the auto industry shared our hesitation over user testing.

[1] Actually, of course, they were right within the circumstances of their study, but most of us can't afford such expensive circumstances.

Low-Cost Dream Car Hits the Road, Pedestrians, Etc.

Four Months from Concept to Reality

Dateline: Detroit Auto giant MegaMotors today introduced Mega Dream 1.0, the first of an exciting new breed of production dream cars. Whereas typical cars are first prototyped, then tested, then redesigned, the MegaDream 1.0 went directly from the drawing board to production in under 5 months. Company spokesperson Clydesdale "Clyde" Snively credited the elimination of iterative design and testing with both the short time to market and a cost reduction in the $40,000 car of almost $300. "Consumers have always admired auto show concept cars. Now they have a chance to actually own one and save some cash to boot!"

We test-drove the new MegaDream and found the car just as exciting as all its Hollywood hype. With its smooth lines and distinctive design, heads are turned wherever you go. In fact, at least one head has to be turned: the driver's. The auto maker boldly attached the steering wheel to the left door, with the driver seated facing said door, "to make driving in reverse just as safe and comfortable as driving forward." (Company steering consultant Edward Harrow suggested they may change this side-saddle arrangement in the second release.)

The lack of user testing led to many of those endless and expensive meetings where groups of engineers close to the project sit around arguing over what naïve drivers will actually do, in the face of no evidence whatsoever. Arguments over the layout of the cockpit area were finally laid to rest by enabling the driver to decide his or her own configuration. Gas, brake, windshield wiper, and other functions are all easily switched around by a driver-preference function of the horn: one beep for brake on the left, two beeps for brake on the right, and so on.

This horn configuration system has so far caused eight fatalities this first week, as drivers in heavy traffic beep their horns to prevent collision, only to find the brake pedal they're stepping on has suddenly become the accelerator. An engineer close to the project admitted they had failed to consider that possibility but said they were planning an interim release that would instead tie the configuration to the current radio station. (No tests of this second design attempt are planned either: "Drivers have no business changing radio stations in heavy traffic, so they shouldn't have any more problems.")

We were impressed with the power of the new car, although the manual spark advance and choke, needed for every gear shift, were a bit daunting. Engineers pointed out, however, that bringing back driver-control over such features increased engine performance and versatility, and none of the engineers were having any problems adjusting them. (Spokesperson Snively admitted that making automatic spark advance a preference item is under consideration for version 2.0, "for the non-power-drivers.")

All in all, we found the MegaDream 1.0 to be a wonderful car, if a bit quirky. Even though it may have its critics, we feel the savings of $300 more than justifies the lack of testing. And for those few drivers who are unhappy with this first release, MegaDream 2.0 will be offered to surviving registered owners for only $36,000, with proof of purchase.

Pretty silly, huh? Meetings where engineers try to guess at how people will react, instead of finding out? Taking major chances at fundamental redesign, like reorienting the driver, with no testing? Assuming that if an engineer can do it, so can the average person?

Guess what, folks? We do it all the time at Apple, and I'm betting you do it too. It always shows, too, but, fortunately, we don't have to do it anymore. At last, we can all test consistently.

Jakob Nielsen of the Technical University of Denmark has recently published a paper entitled "Usability Engineering at a Discount" (Nielsen 1989). In it he refutes the findings of Mantei and Teory and offers a number of useful devices for reducing the cost of user testing to a quite practical level. Since his findings completely confirm the correctness of my own unbiased opinions, I recommend the paper highly.

The common perception of quality user testing is that it involves one-way mirrors, video cameras, and psychology drones in white lab coats, armed with clipboards. Nielsen's research found that usability testing can be done for about 50% of the regular price by eliminating all these elements.

My own experience (Tognazzini, 1986) has been that the chief need for video tape is to produce a propaganda piece that forces recalcitrant engineers to admit their grievous errors. This kind of testing is already expensive, because it implies an on-going war between the testers and the engineers, a war nobody wins, least of all the users.

➤ ***GUIDELINE*** *Inexpensive testing should be done by members of the design team as part of a cooperative effort to produce the best software possible.*

Testing Scenarios and Prototypes

Nielsen recommends developing testing scenarios: Create typical situations that users may face, then build prototypes that will enable exploration of those situations. Two types of prototypes can be built: horizontal and vertical.

➤ ***GUIDELINE*** *Horizontal prototypes display most or all of the full range of the application—menus, windows, dialogs—without going in depth on any one part. Use to test the overall design concepts.*

➤ ***GUIDELINE*** *In areas reflecting new design concepts and technology, build vertical prototypes that carry the user deep into the behaviors of specific parts of the system.*

This enables you to build and test those parts of the system that have potential problems, without having to build a working model of Standard File or some other area that is (relatively) immutable or previously proven.

➤ **GUIDELINE** *Prototypes can and should be created in a matter of days, not weeks or months.*

Prototypes should be built using rapid, inexpensive tools. Nielsen, saint that he is, recommends HyperCard. Some prefer SuperCard, because of its rich supply of standard Macintosh objects; others will prefer Prototyper or other tools of its class. The important thing is that you use a prototyping tool for design.

Test Subjects

Having built the prototypes, it's time to get the test subjects. Now, let's see . . . This is an accounting package, so in order to test whether people can use that new dialog box we will need to try it on Average Accountants. The most average Average American lives in Cincinnati, so we'll fly there and have a marketing research firm arrange a panel of sixty Average Accountants and the video equipment. . . .

Hold it! We are looking for interface problems, not accounting problems. While you want your application and prototypes to be reviewed by one or more accountants, they should be content experts on your team, not random folks in a far-off city. You will also get all the accountancy feedback you can stand during beta testing.

What we want to find, particularly in the early stages, are problems that anyone other than a member of the design team is going to stumble over. The only narrowing of the field I would suggest early-on is deciding whether you want people who are Macintosh-literate, not Macintosh-literate, or people from both groups. Neilsen has found that you need no more than three people per design iteration. My experience bears this out. Any more and you are just confirming problems already found. Better to spend the time iterating the next design.

We have historically used new Apple employees as our subjects. They tend to be a good mix of new and experienced computer users. Colleges are another good source. I have also used many people at computer stores. Stores will often cooperate in allowing you some space for a few hours during a slow time of the week. The test subjects will show up spontaneously, in search of a new computer or this week's hot software. As they come through the door, grab them.

Ten Steps for Conducting a User Observation

➤ **GUIDELINE** *"User observation through thinking out loud" results in our being able to "see inside" our user's conscious minds, to understand what errors in process are taking place.*

So far, things are looking pretty cheap and do-able here: We have created prototypes in a couple of days, and we've gotten one new secretary, one management type, and the guy who drives the sandwich truck to stop by for some testing. It's time to fire up the prototypes and do some testing. To explain how, I will turn you over to Kate Gomoll and Anne Nicol, Apple's resident "Thinking Out Loud" Gurus.

User Observation Through Thinking Out Loud[2] The following instructions guide you through a simple user observation. Remember, this test is not designed as an experiment, so you will not get statistical results. You will, however, see where people have difficulty using your product, and you will be able to use that information to improve it.

These instructions are organized in steps. Under most of the steps, you will find some explanatory text and a bulleted list. The bulleted list contains sample statements that you can read to the participant. (Feel free to modify the statements to suit your product and the situation.)

1. **Introduce yourself.**
2. **Describe the purpose of the observation (in general terms).**

   Set the participant at ease by stressing that you're trying to find problems in the product. For example, you could say:

   - You're helping us by trying out this product in its early stages.
   - We're looking for places where the product may be difficult to use.
   - If you have trouble with some of the tasks, it's the product's fault, not yours. Don't feel bad; that's exactly what we're looking for.
   - If we can locate the trouble spots, then we can go back and improve the product.
   - Remember, it's the product we're testing, not you.

3. **Tell the participant that it's OK to quit at any time.**

   Never leave this step out. Make sure you inform participants that they can quit at any time if they find themselves becoming uncomfortable. This is not only an ethical testing procedure, it is required by law. Participants shouldn't feel like they're locked into completing tasks. Say something like this:

   - Although I don't know of any reason for this to happen, if you should become uncomfortable or find this test objectionable in any way, you are free to quit at any time.

---

[2] K. Gomoll, A. Nicol, "User Observation: Guidelines for Apple Developers," Apple Human Interface Update #11, 1988.

4. Talk about the equipment in the room.

Explain the purpose of each piece of equipment (hardware, software, and so on) and how it will be used in the test.

5. Explain how to "think aloud."

Ask participants to think aloud during the observation, saying what comes to mind as they work. By listening to participants think and plan, you'll be able to examine their expectations for your product, as well as their intentions and their problem-solving strategies. You'll find that listening to users as they work provides you with an enormous amount of useful information that you can get in no other way.

Unfortunately, most people feel awkward or self-conscious about thinking aloud. Explain why you want participants to think aloud, and demonstrate how to do it. For example, you could say:

- We have found that we get a great deal of information from these informal tests if we ask people to think aloud as they work through the exercises.
- It may be a bit awkward at first, but it's really very easy once you get used to it.
- All you have to do is speak your thoughts as you work.
- If you forget to think aloud, I'll remind you to keep talking.
- Would you like me to demonstrate?

6. Explain that you will not provide help.

It is very important that you allow participants to work with your product without any interference or extra help. This is the best way to see how people really interact with the product. For example, if you see a participant begin to have difficulty and you immediately provide an answer, you will lose the most valuable information you can gain from user observation—where users have trouble, and how they figure out what to do.

Of course, there may be situations where you will have to step in and provide assistance, but you should decide what those situations will be before you begin testing. For example, you may decide that you will allow someone to flounder for at least three minutes before you provide assistance. Or you may decide that there is a distinct set of problems you will provide help on.

As a rule of thumb, try not to give your test participants any more information than the true users of your product will have. Here are some things you can say to the participant:

- As you're working through the exercises, I won't be able to provide help or answer questions. This is because we want to create the most realistic situation possible.
- Even though I won't be able to answer your questions, please ask them anyway. It's very important that I hear all your questions and comments.
- When you've finished all the exercises, I'll answer any questions you still have.

7. Describe the tasks and introduce the product.

Explain what the participant should do first, second, third. . . . Give the participant written instructions for the tasks.

Important: If you need to demonstrate your prototypes before the user observation begins, be sure you don't demonstrate something you're trying to test. (For example, if you want to know whether users can figure out how to use certain tools, don't show them how to use the tools before the test.)

8. Ask if there are any questions before you start; then begin the observation.

9. Conclude the observation.

When the test is over:

- Explain what you were trying to find out during the test
- Answer any remaining questions the participant may have
- Discuss any interesting behaviors you would like the participant to explain.

10. Use the results.

As you observe, you will see users doing things you never expected them to do. When you see participants making mistakes, your first instinct may be to blame the mistakes on the participant's inexperience or lack of intelligence. This is the wrong focus to take. The purpose of observing users is to see what parts of your product might be difficult or ineffective. Therefore, if you see a participant struggling or making mistakes, you should attribute the difficulties to faulty software design, *not* to the participant.

Review all your notes carefully and thoroughly. Look for places where participants had trouble, and see if you can determine how your product could be changed to alleviate the problems. Look for patterns in the participants' behavior that might tell you whether the product is understood correctly.

It's a good idea to keep a record of what you found out during the test. That way, you'll have documentation to support your design decisions and you'll be able to see trends in users' behavior. After you've examined the results, fix the problems you found and test the product again. By testing your product more than once, you'll see how your changes affect users' performance.

Thank you, Anne and Kate.

The only thing left to do is to modify problem areas and retest—iteration. Most problems will disappear by the second or third iteration.

This all seems trivially easy, doesn't it. Guess what? It is. This is how Larry Tesler, vice president of Advanced Products at Apple, carried out the extensive testing that resulted in the Lisa computer:

> I've been personally involved in cheap user testing for about 28 years, including my tests for the Lisa. The equipment was me, a Lisa, two chairs, a note pad, and a pen. The subjects were right out of employee orientation. The tests ran one hour. I ran 3 to 5 subjects on each version of each application. I spent another few hours analyzing the notes and writing the report. Cheap. Effective.

How Testing Saves Time

Even when good horizontal and vertical prototyping and testing has been performed, you still need to test the whole application as it comes together in late alpha or early beta, due to Tognazzini's paradox:

➤ ***GUIDELINE*** *Areas of an application where problems are expected have none, while areas known to be perfect are fatally flawed.*

Even though such late human interface testing sends chills up the programmer's spines, it does pay off in eventual time savings.

I experienced my most pathological example of Tognazzini's paradox while Dave Eisenberg[3] and I were working on "Apple Presents . . . Apple" back in 1979, the first time an in-box tutorial had been written for a micro. We had earmarked certain sections of the program as being hopelessly difficult, while others were hardly worth testing. Test subjects found most of the "hopelessly difficult" sections perfectly easy, while the one area we knew would have no problems at all proved fantastically difficult:

Problem:	In "Apple Presents . . . Apple, An Introduction to the Apple II Plus Computer," find out if the user is working with a color monitor.
User profiles:	New owner, customer in a computer store, or member of a class learning to use Apple computers.
Test user profiles:	Customers in a computer store, non-computerists in a classroom environment, friends, and relatives.
First design:	A color graphic would be displayed.
PROMPT:	"Are you using a color TV on the Apple?"
ANTICIPATED PROBLEM:	Those who were using a monochrome monitor in a classroom or computer store situation wouldn't know whether the monitor was black and white or was color with the color turned off. We reiterated the design.
Second iteration:	A color graphic was displayed.
PROMPT:	"Is the picture above in color?"
FAILURE RATE:	25%
REASON:	As anticipated, but incorrectly overcome, those seeing black and white thought their color might be turned down. They didn't answer the question wrong; they turned around and asked one of the authors whether the monitor in question was color or not. A decision was made that the authors could not be shipped with each disk.

[3] For those of you stretching back that far in time, Dave referred to himself in that era as "JDEisenberg." The lack of spaces between his initials led a great many users to conclude there was a problem with their computers. This amused Dave no end.

Third iteration: A smaller graphic with color names, each in its own vivid color was substituted:
GREEN BLUE ORANGE MAGENTA

PROMPT: Are the words above in color?"

FAILURE RATE: color TV users: none
black and white monitor users: none
green-screen monitor users: 100%

REASON: Yes, well, you see, we hadn't exactly thought about monochrome monitors with nice, bright green text. After all, who could have predicted that users might actually think green was a color? Silly twits! Actually, we were extremely fortunate that we accidentally got hold of a prototype green-screen monitor that day. Otherwise, we might have shipped the software with a fatal design flaw.

Fourth iteration: The graphic remained the same.

PROMPT: "Are the words above in more than one color?"

FAILURE RATE: color TV users: none
black and white monitor users: 20%
green-screen monitor users: 50%

REASONS: The black and white monitor users who answered incorrectly admitted that they did so on purpose. (Our methods for wringing their confessions shall remain proprietary.) 50% of the green-screen folk considered that they were looking at both black and green—two colors—and answered the question accordingly.

Fifth iteration: Same display of graphic and colored text

PROMPT: "Are the words above in several different colors?"

FAILURE RATE: color TV users: none
black and white monitor users: 20%
green-screen monitor users: 25%

REASONS: By this time, the authors were prepared to supply everyone who bought an Apple II with a free color monitor, just so we would not have to ask the question. It turns out that around 20% of the people were not really reading the question. They were responding to the question "Are the words above several different colors?"

Sixth iteration: Same display of graphic and colored text

PROMPT: "Do the words above appear in several different colors?"

FAILURE RATE: none

This was a highly interactive tutorial typically taking twenty minutes to complete. This was the only interface issue that required more than one iteration to correct. No matter how many engineers we had crowded into a room to discuss with what areas users were or were not going to have trouble, we would have never hit upon this as the major problem in the application. Had we not tested, we would have had a disaster on our hands: Instead of users having a wonderful first experience, they would have walked away thinking both they and our computer were awfully stupid.

My experience with this and other applications and systems have proven to me beyond a shadow of a doubt that testing can *save* time, rather than cost time because I don't have to work on things that aren't broken. I can instead concentrate my design talents where they will do the most good, on the things that actually need design work. And I don't have to release version 1.01 a week after ship date because of some major human interface problem.

I again put these methods to good use in a later experiment, captured in Chapter 36, "Case Study: One-Or-More Buttons."