December, 1999

Editor's View

Designing for Users

A new book focuses on interaction that accomplishes the user's goals without forcing her to know how it's done

By Tamar E. Granor, Editor

I like computers. I really do. Except when I hate them. That sums up the attitude of pretty much every programmer I know. As for most of the rest of the world, we can just leave off the first two sentences.

Something is very wrong in the world of computing and, in fact, in the world of all high tech products. We love our VCRs, except that we can't program them. We love our microwaves, except when they render our pizza unchewable. We love our cars' keyless entry systems except when they embarrass the heck out of us by whooping like mad in the mall parking lot. What's wrong here? Why can't we have devices that do great things without the down side?

That's the subject of a new book by Alan Cooper, one of the most respected names in the user interface world. I mentioned Cooper's last book, *About Face*, in this space in November '96. That book made me look very hard at the interface of every program I used and every application I designed. The new one, *The Inmates are Running the Asylum*, has me looking at the whole world differently.

Cooper's thesis is, essentially, that programmers and engineers are among the minority of people in the world who look at things in terms of the way they're implemented. Most of the people who need to use those things couldn't care less how they're implemented. They have tasks they want to accomplish and are looking for tools (whether physical or software) that left them do those tasks.

Most mechanical devices are designed to hide their implementations and focus only on the task at hand. A car is a good example. (Thanks to my friend, Barbara Peisch, for pointing this out.) To make a car go, you simply press the gas pedal. You don't worry about sending gas to each cylinder each time it needs more because you don't care *how* the engine works. If the entire mechanism changes (for example, from gas-powered to solar-powered or even doughnut-powered), you still operate the car by pressing the pedal. The implementation is hidden.

Programmers like to know how things work. I know this is true for me, both professionally and privately. My husband and I tend to go to community and even social events and sit there discussing how they're being run and how we think they could be run better. We wonder about how things work and speculate on the reasons why they work the way they do.

This kind of thinking is really important for writing good code. But it doesn't make for good user interaction. That is, the code under the hood has to be designed from clear thinking processes that focus on how things work and how they should work.

But users don't need to see that. Users don't care. Okay, most users don't care. A small percentage of users are people like us who want to know. But that's not who applications should be designed for, just as cars aren't designed for the few people who want to know when the gas is injected into the engine.

Cooper's book is filled with examples of both software and other products (clock radios, cameras, car keys, and more) that are far less usable than they should be because of bad design, much of it design that exposes the implementation. Once he's made the point, he then describes a methodology that results in truly usable products. The examples he gives are really elegant.

One key factor is that the programmers who implement the system are not the right people to design the interactions. In addition, Cooper's approach involves designing for specific users with defined needs ("personas") rather than generic users with fuzzy goals.

For example, roll-aboard suitcases with retractable handles were designed specifically for flight crews. The needs of the flying public were not considered. The result was a very cleanly designed product with wide appeal. Had the luggage going through the usual software design process, it would have been stuck with many more goals and much less likelihood of success.

On the software front, Cooper describes his company's involvement with the software for Logitech's Scanman. For this project, they identified three personas with different reasons for using the product. Cooper's team realized that none of these users was interested in dealing with the Windows file system hierarchy, so the software handles that. Nor did they want to worry about managing the scanner hardware. The one thing all three really needed from the accompanying software was a world-class cropping tool. So rather than building complex software with lots of features, they put the major effort into that one area. Even before the programmers got there, though, Cooper's design team worked hard to find a cropping paradigm that made it easy for users to crop, resize and reorient images. The result was software that surprised Logitech's tech support department by reducing its burden.

As with Cooper's earlier book, my reactions to this one were strong. There were times when I disagreed almost violently, but on further reflection, realized I was exactly the person he was talking about. At other times, the sense of truth was so immediate, I wanted to read the passage aloud to anyone who would listen.

If you're serious about software development, you must read this book. Not to do so would be a disservice to yourself and the people who use the software you write or design. Cooper's lesson is one the software world must learn.