The "Try it and See" Approach to Programming

BY MICHAEL NORTON

I was recently at the University of Nevada in Reno working with Annelle Oelerking and Joyce Farnsworth to put their legacy student information system on the Web using SofTouch's CrossPlex product. Since Reno is the central information systems facility for all six of Nevada's state universities, the task involved setting up six production CICS regions. These regions are accessible to the application programmers through a menu that allows the user to select the desired region. Dick Bond, the systems analyst/programmer who was responsible for setting up the regions, called to inform us that the regions were ready. Annelle exited to the region selection menu but the new regions were not listed, which she noted to Dick. She then noticed that PF8 on the key list was assigned to the scroll forward function. At this point she asked Dick, "What will happen if I scroll forward?" Dick responded, "I don't know. Try it and see."

Later, Joyce, who had helped introduce computers into Reno's public schools, was ruminating on the difference between the way children and adults approach computers, observing that children were much more likely than adults to simply start punching buttons to see what would happen.

Where does the reluctance to simply try something come from? Certainly one explanation is that many people are still intimidated by computers,terrified of messing something up. Yet even an experienced, competent technician like Annelle will almost instinctively ask before attempting even the most rudimentary operation. Why?

In one sense this phenomena is a result of our training. I would imagine that universities still teach programming by having users write programs and submit them to their professors. Most programmers coming out of universities are still meticulous about reviewing their code before they submit it for compile, much less execution. More experienced programmers learn to let the compiler do much of the proofreading. And really burned-out programmers let the operating system isolate their wayward pointers, figuring the system will let them know if there is an access violation.

At one time exhaustive proofreading of programs was standard operating procedure. Computer time was simply too precious and too expensive to take the "try it and see" approach to programming.

Recently, one of our sales representatives was asked by a prospective customer to demonstrate SofTouch System's CrossPlex product. At this point, it was necessary for me to code a script using CrossPlex's scripting language. Although the scripting language is quite simple, I dreaded the thought that the effort would be blind. I had limited access to the site's system and no means of testing the program before the demonstration. It would either work or I would spend an hour on the phone debugging under the intense scrutiny of prospective customers. The situation made me check, double check and triple check the code, and have others fluent in the scripting language look at it as well. We found a couple of syntax errors that we quickly repaired. To my amazement, the code ran flawlessly the first time.

At one time exhaustive proofreading of programs was standard operating procedure. Computer time was simply too precious and too expensive to take the "try it and see" approach to programming. Indeed, it was the bottleneck of programming in the earliest computers that led to the most fundamental concept underlying the design of the modern computer: the stored program. At the time, the mechanical computers generally read their programs from punched tape, executing each instruction before advancing the tape to read the next instruction. This approach was simply too slow for electrical computing devices: The speed advantage offered by electronics would be lost as the speed of the computer would be limited to the speed of the mechanical input device. Thus, the ABC and ENIAC, the first electronic computers, were programmed using plugboards — obviously an enormously time consuming process that proved to be the bottleneck to fully exploiting the potential of the ENIAC. Mauchly and Eckert were no doubt discussing precisely this problem when they were joined in 1944 by a distinguished mathematician and intellectual from the Institute for Advanced Studies at Princeton University, John Von Neumann.

In one of the great ironies of computer history, Mauchly, who had upstaged Atanasoff for credit for developing the first electronic computer, was himself upstaged by Von Neumann, who is credited with developing the concept of the stored program. Arguably the most influential paper in the history of computer science is Von Neumann's "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," which outlined the design of what would

soon become known throughout the computing world as the Von Neumann machine. The Von Neumann machine employed the obvious solution to the programming problem: load the program into memory. This solution allowed programmers to forego plugboards and utilize the advantages of punched tape, without the penalty imposed by reading instructions directly from the tape. Simply read the entire program into memory and execute it from core.

This concept is so fundamental to us it is difficult to comprehend a time when it did not exist. And it was such a simple and perfect solution to the problems with the ENIAC it strains credulity to assert that Mauchly and Eckert had not proposed it even before Von Neumann arrived. Indeed, they were already committed to abandoning the ENIAC and building another machine. Since the fundamental objection to the stored program solution would have been the limited memory of the ENIAC, it is quite reasonable to venture that Mauchly and Eckert had already discovered the stored program solution. But it would be Von Neumann who would be fundamental in designing the successor to the ENIAC, the EDVAC.

The stored program concept of course revolutionized programming techniques. For the first time, a program could modify portions of itself during execution. Yes, the access violation was once actually a highly respected, widely used programming technique. The advent of operating systems doomed the technique but not before it had given birth to the concept that computers could be used in their own programming. The expansive development environments available today are descendants of this concept and every programmer who has employed the "try it and see" approach to programming is the beneficiary of this idea. Letting the compiler or operating system catch programming and logic errors, while certainly inelegant from a purist perspective, is a perfectly logical extension of the idea of using the computer to program itself.

Just as asking before trying is a perfectly logical extension of the idea that operations should be carefully planned before implementation, somewhere between impulsiveness and discipline lies a fundamental truth that explains much about the evolution of the computer — or man, for that matter. *ts*

Michael Norton is the network administrator at SofTouch Systems, Oklahoma City, Okla., which provides both mainframe and PC software solutions. He has written mainframe manuals in addition to articles for a number of publications. Michael can be contacted at norton@softouch.com.