Paradigm Shifts: Part II

by Robert S. Craig

Robert S. Craig

For hundreds of years Swiss watchmakers had a virtual monopoly on watchmaking. The industry generated millions of dollars annually, and grew steadily. Then, in the mid 1980s, a young scientist at a Swiss watchmaking company developed the quartz movement. His colleagues' reaction was one of skepticism and disbelief that anyone would want to replace time-honored techniques with some new, cheap technology. This scientist demonstrated his prototype at a local trade show where it was spotted by Japanese watchmakers who quickly licensed the technology. Three years later the Swiss watchmakers were decimated by their competition and the industry hasn't recovered to this day.

The Swiss weren't stupid: They were the victims of a paradigm shift (which one of them created).

We in the M marketplace have been part of the evolution of underlying technologies available to users. Little more than three years ago the idea of building an application to run on a bit-mapped display controlled by a mouse was an academic curiosity for most M programmers. The variety of products and tools available for most visual environments multiplies. We are nearing the adoption of the first ANSI standard specification for platform-independent graphical user interface in any computer language. Yet, most M development shops, managers, and programmers are still using character-cell terminals.

The way the programmer needs to think about how the application interacts with the user is literally being turned inside-out. This is the paradigm shift faced by M. Inside-out thinking means the screen displays a prompt, the user reads and validates some text, the program then processes the input and moves onto the section of code in sequence. Outside-in thinking focuses on event processing, in which the user controls the interface and is free to select any number of actions, which must be processed as they are received by the application.

The new paradigm also affects how the programmer presents information to the user and can significantly affect how the programmer deals with validation and errors. When my wife asked for my help with a computing problem, I wrote the application using a GUI-based M product. I soon discovered that if I displayed a dialogue box with some well-labeled data entry fields and two buttons (OK and Cancel), she found the application easy and intuitive to use and she was less likely to create an error. It took less training time and less code to validate input. She found the application easier to deal with than character-cell applications I had written previously. I found the program more interesting and fun to write.

Reliable, stable, and robust production environments are becoming the norm. It was once sufficient to restore a backup tape and dejournal to recover from a crash; 99 percent availability (or 88 hours of unscheduled downtime) was good enough for most systems. Customer demand requires higher availability (the system is there when I need it) and reliability (and it works the way it's intended). Platform improvements (redundant processors, multipath disks, RAID technology, and more reliable operating systems) and greater dependence on mission-critical production applications mean customers demand up to 99.9 percent availability—less than nine hours of unscheduled downtime. But many M products do not keep pace with this demand.

Although several M vendors now support transaction processing, most applications do not take advantage of this powerful new feature. The developer who can tell the customer, "You won't have to worry about the system encountering any logical corruption due to a system failure," has an extremely powerful message for the MIS director or system manager running a multimillion dollar business on the right M software. It takes the ability to COMMIT a transaction and ROLLBACK the database if an error occurs. With M Technology we have the ability to create applications that can go toe-to-toe with any other production environ-

ment under the most demanding circumstances by combining carefully designed applications that meet the ACID (atomicity, consistency, isolation, durability) test with fault-tolerant hardware and operating systems.

Object-oriented programming systems (OOPS) represent yet another paradigm, rapidly evolving to eventually dominate the software industry. OOPS's promise is its ability to simplify programming by using classes, inheritance, encapsulation, and polymorphism. A few visionaries articulate the importance of this technology to the M community, but most M designers and programmers are blithely ignorant of the major tenets and principles underlying OOPS.

While some far-sighted individuals and companies we know recognize the importance of these new ways to build applications, many still think, "Well, business has been up every year for the past *n* years, there's no reason to believe this won't continue indefinitely." I believe that the companies run by the latter type will be out of business within five years. Any M software developer who isn't actively planning to adopt these new paradigms will discover that increasingly sophisticated customers will reject old, established technologies for all but the most mundane heads-down, data-entry tasks.

People, it's time to wake up and smell the coffee! We, individually and collectively, need to learn more about these new paradigms and incorporate them into our thinking. If you program or if you manage a development staff, read; get some training; come to the 1994 MTA Annual Meeting for tutorials, discussions, and other presentations on the latest developments available in our community and the future others see; in short, do whatever you can to invest in yourself and your organization by getting the greatest possible exposure to these new technologies. If your people ask for training, send them to the MTA Annual Meeting and elsewhere to get it, otherwise you risk losing them to other companies more committed to the professional and personal growth of their employees. If you're a programmer and your management won't offer training opportunities, *run*, don't walk, to the nearest library, bookstore, or local college and find out about how you can apply these new techniques to designing and developing software.

Otherwise, you risk waking up one day, looking at your quartz watch, and realizing firsthand what happened to all those Swiss watchmakers. M

A previous column dealing with paradigm shifts appeared in the *Board Room* pages of the November 1993 issue of *M Computing*. See "Ahoy Mate! Paradigm Shift Dead Ahead!" by Greg Kreis.—**Editor.**

Bob Craig is the manager of product marketing for InterSystems in Cambridge, Massachusetts. He is on the Editorial Board of the M Technology Association, and a former Director.
