RESEARCH ARTICLE | SEPTEMBER 01 1992

Try Something New: Object-Oriented Thinking FREE

Paul F. Dubois

Comput. Phys. 6, 560 (1992)
https://doi.org/10.1063/1.4823110

THE LAST WORD

Try Something New: Object-Oriented Thinking

Paul F. Dubois

I bought one of the first programmable calculators. I then bought an Atari 800 with two disk drives, a tape drive, a device interface, printer, and a new thing called a modem, before these items had become popular. I have worked on a Sun 100 workstation and a CDC STAR. Yes, that's me, a member of that vital part of the industry, the group that buys all the new stuff before it really works.

Similarly, I've "bought" object-oriented programming before it has become really compatible with most of my daily work on supercomputers. Even so, I've learned a lot of "object" lessons, and benefited a great deal just from my interest and a few side projects.

My first, and most important, lesson has been that I should frequently rethink my assumptions. Object-oriented programming helped me understand why the standard top-down, structured-design orthodoxy is not the right guiding principle for software engineering. Like any good heresy, this revelation induces similar thoughts: what else that I "know" might be wrong? Did I discard an idea for an algorithm long ago because it took too much space? Perhaps that amount of space would look trivial today. Do I have to use a compiled language on this project because interpreted ones are too slow? Maybe they aren't now. Do I have to use that supercomputer? Perhaps a workstation will be faster in real time.

My second lesson is to think object-oriented even when I cannot do object-oriented. As a guiding philosophy, this approach has helped me design better software, even if I have ended up writing in Fortran. I've noticed that in discussing design questions for large software projects, the OO-trained people rapidly converge on a common vision while the OO-untrained flounder about.

My third lesson is that the economics of computing are currently very distorted. I've read (alas, even written) papers about making algorithm X go fast on hardware Y. This type of work is important, because computing faster often makes a qualitative, not just a quantitative, change in the entire scientific process. However, I've rarely read how much it costs to write the software, maintain the software, extend the software, modify the software for reuse, rerun jobs because the software had errors, or replace the software because no one understood it any more. The cost of scientists and programmers dwarfs the amount of money spent on computing hardware, but in our literature, we seem to say that only computer time counts. Money aside, this condition devalues our humanity. The point is: object-oriented programming makes a real difference in the life-cycle cost of software.

Another economics lesson: the more expensive the computer, the worse the working environment. Since supercomputing is a tiny fraction of the software market, software productivity tools such as object-oriented languages, graphics and databases arrive slowly for supercomputers. When they are available, their prices are shockingly high. Even when I can use a commercial product, my collaborators at educational institutions have little money to spend on these tools in order to work with me. Likewise, if I write something others can use, there is no institutional framework to assure them of its support, and legal issues and market size make commercialization difficult.

Finally, I've learned that even if I don't use OO-technology, the rest of the computer world will. "Object-orientedness" increasingly pervades the operating systems and software with which I work. An understanding of object-orientedness will become an increasing asset in making the most of my computational environment.

Beyond OOP

Object-oriented programming is just one aspect of an enormous subject: how to program effectively for scientific applications. Beginning next year, *Computers in Physics* will have a new Department entitled "Scientific Programming." As the first editor of this department, I'll try to cover the process of scientific programming and point out some of the interesting ways in which people are coping with change and the coming of parallel, distributed, heterogeneous computing.

Here at Livermore, my team recently completed the porting of a very large Cray application to both the Sun and Hewlett-Packard workstations, where it gets roughly the same answer, and produces dump and post-processing files that are portable without translation. Therefore, I've decided to begin Scientific Programming in the Jan/Feb issue with a discussion of portability. In that issue, I'll also explain more about the topics I hope to cover.

I'd welcome hearing from you about topics of interest. You can write to me at dubois1@llnl.gov or Lawrence Livermore National Laboratory, L-472, Livermore, CA 94550.

Paul Dubois is a Computer Science Team Leader with the Computational Physics Division, Lawrence Livermore National Laboratory, CA 94550.