Abstract

A perennial concern of those who teach about computers is the rapid change that we see in our field. The hardware and software that dominated the field just four years ago are not even considered interesting today. When we think about our students, we must wonder how we can teach anything that will be relevant when they graduate — or five years later.

While the pace of change in computing is unprecedented, the problem of teaching engineers in rapidly changing fields is not. When I received my electrical engineering education in the late 60's, my teachers were as concerned about rapid change as we are today. When I entered engineering, the “real world” was vacuum tubes, and some people told me that semiconductor physics was “useless theory.” The wise members of the faculty recognized that what we taught had to be useful now, as well as in the future.

Today, teaching in an engineering environment once again (after a hiatus of two decades) I can see the wisdom of their choices. The engineering books on my shelves, purchased in the late 50's, are still true and relevant. In contrast, most of the computer books on my shelves dated in the 80's, are out of date, irrelevant, archaic, and in many cases, obviously wrong. Things have changed quickly in both areas, but what I learned two decades ago was fundamental and has remained important.

As long as I have been observing computer science education, there has been concern about keeping up-to-date. The ACM curriculum recommendations are frequently updated and conference discussions are full of the latest buzzwords and references to the latest research papers. The engineering education tradition, however, is different: it stressed focusing on fundamentals and avoiding fads. In contrast, currently, “teaching new research results” is a scathingly negative comment in all but the most advanced undergraduate engineering courses.

In this talk, I will propose and defend a list of fundamental concepts that should be taught in software engineering courses. I will argue that only by not teaching the latest fads, can we give our students an education that will stand them in good stead a decade or two from now.
Biography

David Lorge Parnas is a professor in the Department of Electrical and Computer Engineering at McMaster University in Hamilton, Ontario. He has been a professor at the University of Victoria, the Technische Hochschule Darmstadt, the University of North Carolina at Chapel Hill, Carnegie Mellon University, and the University of Maryland. He has also held non-academic positions advising Philips Computer Industry (Apeldoorn), the United States Naval Research Laboratory in Washington, D.C., and the IBM Federal Systems Division. At NRL, he instigated the Software Cost Reduction (A-7) Project, which developed and applied software technology to aircraft weapon systems. In addition, he has advised the Atomic Energy Control Board of Canada in the use of safety-critical, real-time software at the Darlington Nuclear Generation Station.

Dr. Parnas is interested in most aspects of computer system design and is the author of more than 180 papers and reports. In his teaching, as well as in his research, he strives to find a “middle road” between theory and practice, emphasizing theory that can be applied to improve the quality of our products.

Professor Parnas received his Ph.D. in Electrical Engineering from Carnegie Mellon University, as well as honorary doctorates from the ETH in Zurich and the Catholic University of Louvain. In addition, he won an ACM “Best Paper” Award in 1979 and two “Most Influential Paper” awards from the International Conference of Software Engineering. He is a Fellow of the Royal Society of Canada, a Fellow of the Association for Computing Machinery (ACM), and he is licensed as a Professional Engineer in the Province of Ontario.