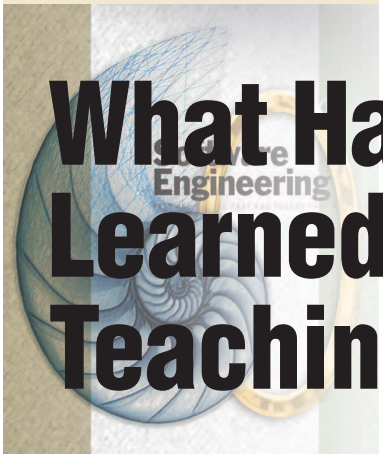

What Have We Not Learned about Teaching Programming?

David Gries
Cornell University

***Technique Recipe**
The height of technical felicity is to combine sublime simplicity with just sufficient ingenuities to show how difficult to do it is.
Piet Hein

When the NATO Conference on Software Engineering was conducted in 1968, academicians and industry people came together practically for the first time to discuss problems in software design and development. Both camps admitted publicly that they did not know how to develop software well. Frequently missed deadlines, cost overruns, buggy software, and the inability to teach programming were clear symptoms of the "software crisis."

CS Professor's Nondilemma
I do so want students to see
beauty and simplicity.
A language used just has to be
one only with that property.
Therefore, and most reasonably,
I will not and do not teach C.
David Gries

That conference and the people who attended it inspired a great deal of research in the late 1960s and 1970s. Three of the most influential and prolific contributors to that research were Edsger W. Dijkstra, Tony Hoare, and Niklaus Wirth. Their contributions included an axiomatic basis for programming languages; a theory of weakest preconditions, which led to a methodology for the formal development of algorithms; stepwise refinement; structured programming; and advances in programming language design with Pascal, guarded commands, and communicating sequential processes.

In a nutshell, these three researchers strongly influenced programming language design, programming methodology, and program correctness. As I look back and ask how well we have absorbed their work and incorporated it into our teaching, I am sorry to say that, to a large extent, we have failed.

These innovators made specific contributions to our profession that are no longer emphasized in our teaching:

- They stressed simplicity and beauty and believed that language shapes the thought and mind, as Benjamin Whorf said. These principles are rarely mentioned now in programming texts; we sold our programming soul when we began teaching the pedagogically unsound and intellectually ugly languages C and C++ to beginners.
- They emphasized methodology—they talked about principles of program development, like stepwise refinement. Yet, most introductory texts are "pro-

Keep Things Simple
Learn not only to simplify
But also not to complify.
David Gries

*Note: Danish poet Piet Hein is known to a wider public for his thousands of short, aphoristic poems called grooks (gruk in Danish). In addition to the example from Hein provided here, the author has created some grooks to accompany this article.

0018-9162/06/$20.00 © 2006 IEEE

gram texts" and not "programming texts" because they say little about the programming process. Instead, they present programs and simply expect students to be able to write similar ones.

- They focused on correctness concerns and the idea that a program and its proof should be developed hand in hand, with the proof ideas leading the way. Yet most graduating computer science majors do not know the meaning of the word *invariant* even though the loop invariant is a prime software engineering tool that allows understanding each independent part of a loop—initialization, termination condition, progressing toward termination—without having to look at the other parts.

Admonition
In correctness concerns
one must be immersed.
To use only testing
is simply accursed.
David Gries

We need to look seriously at how we teach programming. The purpose of an education should not simply be to pour facts into students, but rather to teach them to think. Here, I am reminded of a saying that includes the word *horticulture*: "You can bring a horticulture but you can't make her think."

Well, I believe our students are not that way. I believe that we can teach them how to think about the programming process far more effectively than we do now. I also believe that doing so would not only increase our effectiveness as teachers but also make our entire curriculum more efficient. The topics we need to teach more effectively include correctness concerns; the program development process, not only simplifying but also not "complifying" in the first place; how notation can help or hinder; problem solving; and how to find bugs.

Defining Variables
Know what your variables mean
before you write statements to use them.
Otherwise vagueness ensues,
and your code will only confuse them.
David Gries

The reader will realize that I have used hyperbole in this brief summary to make my points, and I probably have stepped on some toes. I write this to get people to think seriously about how we teach programming—to start a dialogue, a debate.

I have been teaching programming for 40 years, and I still teach the first programming course to more than 100 students each semester. Therefore, I believe I have some right to speak my mind. I use an OO-first approach in Java, with all aspects of OO, including inheritance, casting, and so on, coming before recursion and loops—and I teach the latter two topics from a correctness and developmental point of view.

***David Gries** is associate dean of engineering and a professor of computer science at Cornell University. Contact him at dgries@cs.cornell.edu.*