Form over Function

Teaching Beginners How to Construct Programs
(Distilled Tutorial)

Michael Sperber
Active Group GmbH
michael.sperber@active-group.de

Marcus Crestani
University of Tübingen
crestani@informatik.uni-tuebingen.de

Abstract

Teaching beginners how to construct programs is hard work. This is partly because teaching how do any sufficiently complex activity is hard work. Another part of the problem is that our knowledge about the systematic construction of programs is quite young, and thus our knowledge about the didactics of the discipline is even younger. This makes us question reports of sweeping successes in introductory-programming classes by use of a single device, be it the use of a specific programming language, specific accompanying book, software — or robots. That does not mean the choice of individual devices does not matter. However, designing a successful introductory-programming course is a comprehensive activity, in teachers should be ready to consider and continually question all aspects of the course.

It starts with a clear idea of what we want to teach: We believe that much of programming should be done *systematically*, with a clearly defined path from problem to program. Matthias Felleisen's group pioneered the *Program by Design* approach [3], and we have been following in their footsteps [4]. Everything we do flows from our desire to enable students to construct programs systematically. However, this has been exceedingly difficult: The very idea of programming systematically is controversial among practitioners as well as educators, and the idea of doing *anything* systematically is anathema to many of our students.

In the tutorial, we will try to distill the basic ideas and insights that have driven the development of our introductory course, which has since been adopted by the Universities of Freiburg and Kiel.

In particular, a number of conclusions about effective teaching were not as we had originally expected:

1. Form matters more than working programs.
2. Our students are not like us.
3. The requirements for a teaching language are different than the requirements for a production language.
4. Students have to like neither us nor Scheme to learn successfully.
5. Telling students that what they are learning is worthwhile does not matter.
6. Personal supervision is important.
7. Traditional teaching evaluations are almost worthless.
8. Continual evaluation of teaching success is important.
9. Grade pressure works.
10. Cheating is a significant problem.

Despite #3, we use a variant of Scheme introduced through a sequence of language levels [1]. We do see Scheme as only a means to an end, however: Just because a language is great does not mean it is great for teaching beginners. The one killer feature that Scheme offers the educator, however, is that it enables us to change the language we present to beginners easily, through macros and other language-defining features of the Racket programming environment (formerly DrScheme) [2]. In fact, in the early phases of developing our course, we changed the language on an almost weekly basis as we observed that students were having trouble with individual aspects of it. Over the years, our languages have diverged more and more from the Scheme professional programmers use, and we expect that trend to continue in the long run.

ACM SIGPLAN Workshop on Scheme and Functional Programming September 9, 2012, Copenhagen, Denmark
Copyright © 2012 ACM [to be supplied]. . . $10.00

Categories and Subject Descriptors D.2.10 [*Software Engineering*]: Design—Methodologies; K.3.2 [*Computers and Education*]: Computer and Information Science Education—Computer Science Education

General Terms Design, Languages

Keywords Introductory Programming

References

[1] M. Crestani and M. Sperber. Growing programming languages for beginning students. In S. Weirich, editor, *Proceedings International Conference on Functional Programming 2010*, Baltimore, Maryland, USA, Sept. 2010. ACM Press, New York.

[2] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. The DrScheme project: An overview. *SIGPLAN Notices*, 33(6):17–23, June 1998.

[3] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. *How to Design Programs*. MIT Press, 2001.

[4] H. Klaeren and M. Sperber. *Die Macht der Abstraktion*. Teubner Verlag, 1st edition, 2007.