Taking Part-Time Programmers Seriously

Jesse A. Tov
Northeastern University, Boston, Mass., USA
tov@ccs.neu.edu

Elizabeth Tov
Boston College, Chestnut Hill, Mass., USA
elizabeth.tov.1@bc.edu

Programming is now a necessary activity for people working in many disciplines, from biology to sociology. However, the languages and tools used by many "part-time programmers" have not benefited from programming languages research, which presents both an opportunity and a challenge. Given the large number of new programmers who lack bad habits and prejudices, we have a fresh chance to encourage adoption of robust techniques and good technology. To make this happen, programming languages experts must develop languages and tools that provide demonstrable advantages to new part-time programmers (§1), and we need to invest in a pedagogy that ensures their success in acquiring relevant programming skills (§2).

1. Discipline-Specific Languages

Biologists are now part-time programmers; sociologists, linguists, and economists are programmers too. Physicists have long been programmers, and they have earned a reputation for writing bad code in difficult, error-prone languages. For physics, the train has probably left the station, but in other disciplines, programming languages researchers may have a chance to make a positive difference. We can help, first, by providing appropriate technology and demonstrating its benefits.

Consider, for example, quantitative research in sociology. Sociologists typically use software packages such as SPSS and Stata to perform statistical analyses. Each of these is a graphical program built around a core domain-specific language for statistics. Users of these programs may begin by using them in a menu-driven manner, but many eventually advance to typing commands into an interactive interpreter. Users may save a sequence of commands in a "script" in order to re-run an analysis again in the future, often modifying the script before each subsequent run to reflect changing circumstances or strategies. This suggests a need for parametrization, but in practice the poor abstraction facilities provided by these languages are left unused by all but the most expert users. (Even iteration, which programmers would consider indispensable for data management, is not widely understood.) Better statistical languages, such as R, are available, but these have a reputation among sociologists that the effort to learn them is not worth the payoff.

Programming practice in sociology is thus caught in a gap between antiquated, badly designed languages that no one learns properly and a more reasonable (albeit strange) language that no one learns at all. Yet quantitative sociologists would clearly benefit from both improved tools and improved programming skills. Besides a language with statistical facilities, this will require an IDE with a graphical data editor, the ability to perform simple statistical analyses using menus (while generating syntax), and convenience features such as identifier completion. However, developing a superior tool is insufficient. We also need to demonstrate to potential users why adopting new technology is worthwhile, and to provide a way for them to learn to use the new tools.

2. How to Design *X* Programs

If we build it, will they come? Not without our help.

The undergraduate social science major in a required introductory statistics course performs two marginally related tasks. First, she memorizes mathematical formulae for several statistics; second, she uses a software tool, such as SPSS, to actually perform statistical analyses. Much of the course is about learning to use the software, but any understanding of how the software connects to the memorized formulae is cursory, and understanding of the software itself is shallow. We propose that programming would facilitate deeper and more integrated understanding.

The *How to Design Programs* curriculum has been used successfully to teach introductory programming to thousands of high school and college students at a variety of skill levels. It assumes no prior programming knowledge and, importantly, no specialized domain knowledge either. Instead, the domain is pictures and time: Students learn to create interactive animations and games in an untyped, functional programming language designed specifically for teaching. We envision a course where the domain is, instead, statistics—that is, we propose making the introductory course in statistics a course in programming, with statistics as the domain for examples and exercises. Students who implement the basic statistical operations will find both the mathematics and the software less mysterious, and the programming skills acquired will translate to more effective use of other statistics languages in the future.