

PL Research and Its Consequences on PL Curriculum

James Larus
Microsoft Research

SIGPLAN Workshop on Undergraduate Programming Language Curriculum
Harvard University
May 29-30, 2008

I write this contribution as a friend, though not a fan, of programming language research. In my research—ranging across computer architecture, compilers, tools, software development, and parallelism—I have repeatedly seen that programming languages can both enhance and retard other, more universal goals, such as improving software production and quality.

The programming language research community—a fraction of the much larger and more vibrant language community—has increasingly turned inward and focused on narrow, technical contributions rather than the larger concerns of the computer science community or programmers in general.¹ It is not surprising that our colleagues and students reciprocate our lack of interest in their problems by replacing language courses with newer, more exciting offerings that address broader issues. Bluntly, the PL research community brought this workshop on itself.

I believe this situation is reversible. The larger, non-academic programming languages community is thriving, and it is widely credited with providing essential tools for revolutionary infrastructure like the Web. In the early days, enormous numbers of web sites were written in perl, the “duct tape of the Internet.” Everyone recognized that it was an ugly, ramshackle language, but it was dynamic and had excellent support for text manipulation. Its successors, Python and Ruby, are improvements in many respects, but, like perl, they were developed outside the PL research community, which has paid them little attention.² Why did the PL research community miss this opportunity to improve programming in this new domain, which offers challenging problems, a complete lack of legacy code, and world-wide impact? Why did the community focus its attention on Java, a language only marginally different from previous languages, instead of looking at the underlying problems in this new world of programming? It is conservative, incremental decisions like this that give the field of programming languages its dusty, out-of-touch air.

Programming in general is difficult, and new domains, such as web, distributed systems, Multicore, etc., present new and difficult programming challenges. The key insight is that in all of these areas an appropriate programming language facilitates writing software. This is the magic of programming languages: deep down, they are all equivalent, but a well-designed, well-chosen language can make an enormous difference in programmer productivity and program quality. This magic has been

¹ My apologies, in advance, to everyone this observation mischaracterizes.

² Perl is mentioned in 7 POPL papers, only one of which actually discusses the language (Audrey Tang, POPL 07). Python is mentioned in 6 POPL papers, none of which focuses on the language. Ruby is mentioned in 5 POPL papers, none of which focuses on the language.

demonstrated again and again, and is widely believed. Language discussions are heated and intractable, because programmers feel a bond to the tools that let them express themselves and make them successful. This is the bedrock truth for programming languages; they have deep connections to their users that are not shared by other research disciplines, such as algorithms, compilers, or computer architecture. Why is this connection not born and fostered in undergraduate PL courses?

Applying languages to difficult problems used to be the focus of the languages community. Consider the titles of early SIGPLAN Symposia:

- Extensible Languages, 1971
- Programming Languages for Parallel & Vector Machines, 1975
- Language Design for Reliable Software, 1977
- AI and Programming Languages, 1977
- Data Abstraction, 1980
- Text Manipulation, 1981
- ADA, 1980
- Programming Languages and System Software, 1983

Compare these topics to the papers in POPL or PLDI today. As the tools of languages community grew more precise and mathematical, their problems grew smaller, and this magic connection was lost. It is time to rediscover it. Here are a few tangible suggestions how to achieve this goal:

- Start a general discussion about the direction and techniques of the field.³
- Reach out to the larger language community by inviting them to submit papers and give keynotes at major conferences and workshops.
- Accept papers on interesting new languages, even if they do not contain a full, formal semantics.
- Find programming language papers that are published in other fields' conferences and commercial venues, and make sure that your PL graduate students read them.
- Organize workshops and participate in conferences that cross discipline boundaries.
- Work with researchers and commercial developers to develop and apply languages to important problems.

And, back to the topic of this workshop, teach undergraduate programming languages as the fundamental basis for programming, not as a mathematical discipline. Inelegant languages (perl, python, Ruby), imprecisely formulated concepts (programming patterns), new domains (robotics, web programming, parallelism) are the world that your students will graduate into. Making the connection between programming languages and programming does not diminish PL, but instead emphasizes its central position in the undergraduate education.

³ A model is the computer architecture community's concern that excessive reliance on simulation made it impossible to publish new, big ideas in conferences full of exhaustively simulated incremental advances: K. Skadron, M. Martonosi, D. I. August, M. D. Hill, D. J. Lilja, and V. S. Pai, "Challenges in Computer Architecture Evaluation," *IEEE Computer*, vol. 36, pp. 30-36, August 2003.