

PROGRAMMING LANGUAGES MATTERS

Richard Wyatt
Department of Computer Science
401 Anderson Hall, West Chester University
West Chester, Pennsylvania 19383
(610) 436-3230
rwyatt@wcupa.edu

Lonnie Fairchild
Department of Computer Science
101 Broad Street Plattsburgh, New York 12901
SUNY Plattsburgh, Plattsburgh, NY
(518) 564-2783 lonnie.fairchild@plattsburgh.edu

David Hemmendinger
Dept of Computer Science
Union College
Schenectady, NY 12308
(518) 388-6270
hemmendd@union.edu

Adrian Ionescu
Department of Mathematics and Computer Science
Wagner College 10301
One Campus Road, Staten Island, New York 10301
(718) 390-3245
ionescu@wagner.edu

Amruth N. Kumar
Department of Mathematics and Computer Science
Ramapo College of New Jersey
505, Ramapo Valley Road, Mahwah, NJ 07430-1680
(201) 684 7712
amruth@ramapo.edu

SOME MOTIVATING QUESTIONS AND ISSUES

What, I asked the panelists, should an undergraduate Programming Languages course at a mid-level institution consist of? ABET and CC-2005 are not much help. They leave an enormous range of options; all they say on the matter is:

ABET: IV-6. The core materials must provide basic coverage of ..., concepts of programming languages... [2008-2009 Criteria for Accrediting Computing Programs, p.3]

ACM/IEEE: Theory of Programming Languages Principles and design of programming languages including grammars (syntax), semantics, type systems, and various language models (e.g., declarative, functional, procedural, and object-oriented). [ACM and IEEE Computing Curricula 2005, p.56]

Text books, though they provide a wealth of material, present such a range of material that it is pretty wide open as to what should be the smallish subset to include in a one semester undergraduate course. More importantly, they vary enormously: some focus on conceptual matters only, some on implementation matters, some on merely comparing programming language, and so on.

Although the panelists will not, of course, be restricted to them alone, the following questions and issues are those which motivated the panel's formation:

- (a) Should a Programming Languages course be the primary vehicle for satisfying any, some, or all of the following ABET criteria:
 - concepts of programming languages?
 - principles and design of programming languages?
 - exposure to various language models (e.g., declarative, functional, procedural, and object-oriented)?
 - exposure to various languages?
- (b) What, of the large number of topics that could be included:
 - are central and must be included?
 - could reasonably be included?
 - probably should not be included (because they are too difficult, or they are insufficiently central, or there is insufficient time, etc.)?
- (c) Should the focus be on any, some, or all of:
 - conceptual matters?
 - comparative programming languages?
 - implementation issues?
 - exposure to languages from different paradigms?

- (d) Should an upper division programming languages course actually teach undergraduates how to program in some of the less familiar languages such as Prolog or Haskell? If so, how much class time should be spent on it? Alternatively, perhaps students should be able to pick up new languages entirely on their own? What if the language is in a new and unfamiliar paradigm?
- (e) How many programming languages, and from which paradigms, should a student be reasonably proficient at and how many should they have a passing acquaintance with:
 - before they take a programming language course?
 - by the time they graduate?
- (f) What are the reactions of students familiar with C++ and/or Java when they encounter Lisp, ML, Haskell, Prolog, etc., which they probably have not heard of and may think are irrelevant in today’s computer science world. (Or, to put it less delicately – how do we react if a student thinks: why do I have to learn some odd programming language this professor liked 30 years ago (or last week) and is now dead and gone (or no one will ever use)?)
- (g) With the rise of IT, some have wondered about the long term prospects of conventional Computer Science at the kind of mid-level institution typical of CSSCNE attendees. In such an environment (and the ABET and CC-2005 views notwithstanding) is a programming languages course still relevant, needed, worthwhile, or (even) essential?
- (h) Can a programming language course maintain sufficient popularity with students to allow it to continue to be offered in an environment in which administrators are increasingly cutting back on courses with smallish numbers of students? In the light of such pressures, is making the course a required one a good solution to dwindling enrollment? Can we make the case that it is of such importance that it needs to be offered even if enrollments are low?

ABSTRACTS

- **Lonnie Fairchild**

One of the commonly expected outcomes of the programming languages [PL] course is to provide students with the confidence and understanding to learn new languages, and maybe even new paradigms, on their own. This is also important for computer science students who may not take a PL course. I have tried to achieve this when teaching the PL course by using an ongoing project in which the implementation language is shifted several times. At each shift, the students are given as a starting point a solution of the completed portion in the new language. I will discuss some lessons learned with this approach and consider how the goal can be achieved outside of a PL course.

- **David Hemmendinger**

When the ACM Interim Review of the CS 2001 curricular guidelines called for comments, those on programming-language issues vastly outnumbered comments on all other topics. The draft review document says that “programming language issues will be central discussion items in the next round of curriculum revisions.”

It is puzzling that the 2001 guidelines so downplayed programming-language issues that there appeared to be no urgent need to have a course devoted to them. It is a truism that learning a foreign natural language teaches one about one’s first language, and the same applies to computing. Studying programming languages, whether in a course that focuses on conceptual matters, implementation issues, or language comparisons, teaches students about the tools that they use. Solving problems with a language quite different in style from the prevailing imperative and object-oriented ones, whether it be a functional or a logic language, teaches new ways to think. Today, when a major challenge facing computer science is to find the abstractions that enable us to write parallel programs well, we need to educate students in how languages work more than ever.

- **Adrian Ionescu** CS 356, Programming Languages, is a course offered in the CS program at Wagner. It is the middle part of a three-course sequence, which starts with CS 325, Theory of Computation, and ends with CS 456, Compilers. They are connected by the following sequence of ideas. We start with Theory of Computation going over regular languages and context-free grammars for about 2/3 of the semester. In the following course, Programming Languages, we emphasize the Programming Language Syntax: we spend close to a month reviewing regular languages and CFG and the way they are applied to scanners and LL parsing (very brief introduction to LR parsing). The last course in the sequence, Compilers, is taking over by having a thorough discussion of LL and LR parsing with several projects involving flex and bison (former lex/yacc). We use the three different textbooks which are perfectly integrated in terms of notation and concepts. In Programming Languages, in addition to some traditional concepts (syntax analysis, scope, binding, semantic analysis, control flow, data types, control abstraction, etc.), for about two weeks at the end of semester we also discuss some of the Functional and Logic Languages concepts, e.g., Lisp/Scheme and Prolog. There is no intention of making students proficient in either language, just give them the flavor of a different “way” of programming (they are all familiar with C++ and Java). (We also have a new Information Systems Major, and the new major does not require the three-course sequence.)

- **Amruth Kumar**

Our Programming Languages course serves as the primary vehicle for discussing the similarities and differences among the designs of various programming languages, the desirability of language features, language implementation issues and the most significant programming paradigms. It is a required course often taken by students in their

senior year. They are expected to be proficient in object-oriented programming in C++. In the course, students learn for a week each, functional programming in (pure) LISP, logic programming in Prolog and event-driven paradigm in Java. They are expected to implement the same game (e.g., Solitaire, Battleship) in all four languages: C++, LISP, Prolog and Java. They are not allowed to use imperative constructs or side-effects in LISP and Prolog projects.

A common complaint among students is about the relevance of studying languages such as LISP and Prolog. I tell them that these languages are like cross-training for the mind - learning to think differently. One of the best compliments I have received from an alumnus was on why he thought this was the course he liked best - it made me do things I never thought I could.

- **Richard Wyatt**

In preparation for ABET evaluation, the programming languages course at my university has just been made a required course, beginning this semester. Students are approaching the course with noticeably more seriousness; I do not think it a coincidence. A few years ago students seemed quite unenthusiastic about the course: I even heard one refer to it as the “Unimportant Programming Languages” course (at the time it was called “Non-Imperative Programming Languages”). I have since made much of the importance, relevance, and actual uses to which they are put, of the language covered. A couple of papers by Paul Graham have proved most useful.

I provide a little language instruction, just a few classes to get the students “rolling”, in Lisp, ML and Prolog. A knowledge of C++ and/or Java is assumed. The rest of the course is on conceptual matters. We do not discuss implementation in this course. Indeed, I make the point that, as with any technology, programming languages are increasingly trending towards being able to be used without any knowledge of the underlying workings. There are, of course, several programming assignments.

In addition to the more or less standard topics, we also cover evaluation order, the Von Neumann and lambda calculus models of computing, and the weaknesses of imperative programming languages.

BRIEF BIOGRAPHIES

Lonnie Fairchild teaches computer science at SUNY Plattsburgh. She has regularly taught the programming languages course. Other teaching has included courses in introductory programming, theory of computation, artificial intelligence, and compiler development.

David Hemmendinger teaches courses on programming languages, computer architecture, concurrency and parallel programming, and the history of computing. He has written on these areas as well as on the history of science. He was a co-editor of the 2000 edition of the *Encyclopedia of Computer Science*, and has written articles on programming-language

topics for it and for the *Encyclopedia Britannica*. His current research is on programming language history and other areas in computing history.

Adrian Ionescu is a Professor of Mathematics and Computer Science at Wagner College. He received his Ph.D. in Mathematics from Texas A&M University, his M.S. in Computer Science from University of Houston, and his B.S./M.S in Mathematics from University of Bucharest. His research areas include functional analysis, system analysis, numerical analysis, and theory of computation.

Amruth Kumar is Professor of Computer Science at Ramapo College of New Jersey. He has been teaching the Programming Languages course for 15 years. He has developed online software on language implementation issues for the Programming Languages course as well as on programming concepts for Computer Science I (www.problets.org).

Richard Wyatt teach Computer Science at West Chester University in PA. His research interests include Artificial Intelligence, Functional Programming Languages and Philosophical Issues in Computer Science. He has been a member of the CCSCNE Board since 1995 and is the present Board Secretary and previous Board Chair.