Getting Started in Programming Language Design Research

Guy L. Steele Jr.
Sun Microsystems Laboratories
May 9, 2007
Read

• Read everything you can get your hands on
  > I killed time between classes reading back issues of CACM
• Well, actually, be selective: ask for advice
• There really are some “classics”
  > Naur et al., *The Revised Report on ... ALGOL 60*
  > Landin, *The Next 700 Programming Languages*
  > Dijkstra, *Go To Statement Considered Harmful*
  > Knuth, *Structured Programming with Go To Statements*
  > Milner, *A Proposal for Standard ML*
Attend Conferences/Read the Papers

- Principles of Programming Languages (POPL)
- Programming Language Design and Implementation (PLDI)
- Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)
- Principles and Practice of Parallel Programming (PPoPP)
- Architectural Support for Programming Languages and Operating Systems (ASPLOS)
- International Conference on Functional Programming (ICFP)
- Languages, Compilers, and Tools for Embedded Systems (LCTES)
- International Symposium on Memory Management (ISMM)
- Virtual Execution Environments (VEE)
Learn and Use Several Languages

• Pick languages very different from each other
  > Say: Java, Lisp, ZPL, Haskell, Fortran, COBOL, and APT
    > These span wide ranges of time and application areas
    > They span a wide range of styles and mechanisms
    > Some are trendy and some seem downright fuddy-duddy: why?

• Write substantial amounts of code in each
  > Try to use them idiomatically
  > Don't just write “Fortran with Lisp syntax” (or vice versa)

• Write the same program in several languages
  > What becomes easier? What becomes harder?
Design New Features or Languages

• Try Biggle's “Rule of One”: what one improvement would you make to your favorite language?
  > Don't go for the shiny object: think carefully

• Try to make your new features fit the existing style
  > So what is the “existing style”?

• Implement your new feature!
  > Join a project, or find an open-source implementation
  > Or build a “toy interpreter”

• Design and implement an entire “toy language”
  > Which features are really essential?
  > Which are “merely” conveniences?
Read Implementations

• Read the code! Read a library! Read a compiler!
  > Is it elegant? Is it yucky? What would you do differently?

• For that matter, read applications code, too
  > Is it yucky?
    > If it is, was that the coder's fault or the language designer's fault?

• Would the implementation be better or worse if it were coded in the language being implemented?
  > Is self-implementation a goal of the language design?
Question Everything

• Not to be adversarial, but to understand why
  > Why use “=” for assignment?
  > Why worry about the difference between “1” and “1.0”?
  > Why evaluate argument expressions before a call?
  > Why declare variables before they are used?
  > Why declare statement labels before they are used?
  > Why call “free” after using a malloc’d structure?

• For every feature of your favorite language, try to find another language that does it differently
  > Then try to figure out why
  > Was it a good reason? Is it still a good reason today?
Know Your History

• Stuff we now take for granted had origins
  > Before acceptance, lots of variants were explored
  > Examples: if-then-else, case, records

• We can learn from the experience of others

• It's only about 50 years' worth, not 500!
Use the Internet

- Two decades ago, I'd have said “Use the Library”
- Google and (especially) Citeseer are your friends
  - Online databases have forward references!
- Online ACM Portal is a great resource
- Save yourself some embarrassment
  - Find relevant work
  - Dig up and verify citations for yourself
    - Citations often have errors
    - While you're at it, read the cited works!
Polish Your Natural Language Skills

• Learn one or more foreign languages
  > Learn how human communication works
  > Programming languages are human communication, too

• Work on your writing skills
  > Strive for clarity and consistency
    > Watch out for slippery pronouns
    > Try not to use a word in two senses
    > Read with a critical eye: could this sentence have two meanings?
  > Learn the mechanics of your natural language
    > Read style manuals
    > Know when you don't know—then look it up!
My Desk