The Origin OF THE
Breakdown OF THE

Larry Wall

Approximately 12 years ago, the Unix programming universe consisted of two linguistic cultures. You either programmed in C, or you programmed in the shell (for some value of shell). The two systems were good for different things, so their capabilities were widely viewed as complementary. The revelation that came to me one day was simply that these capabilities were not, in fact, opposite ends of a single dimension, but rather the axes of a two-dimensional graph.

The C programming language was good at getting down into the "innards" of things, but wasn't very good at whipping things up quickly. Whereas the shell was good at whipping things up quickly, but couldn't get down into the nitty-gritty stuff. So if those were the two dimensions of the graph, then each language stayed near its own axis, and there was this big blank area in the middle.

When I decided to fill that blank area, it occurred to me that I should design the language I wanted. Not that I could help it—I am a synthesist at heart, and could hardly help borrowing principles from everything I've ever dabbled in: linguistics, music, science, missiology, etc. I like to tell people that Perl is composed of equal parts computer science, linguistics, art, and common sense. (Note that computer science is outnumbered three-to-one.)

Traditional computer languages have mostly tried for minimalism. They attempt to express in the fewest number of features everything that users might want to do. But linguistics, art, and common sense conspire to tell us that people would rather spend a little effort learning a richer, more expressive language, especially if they can learn as they go. Human languages are not minimalistic. That should tell us something.

The computer languages taught in schools and used in industry, from Fortran and Pascal to C and C++, all share in common the mis-feature of forcing programmers to worry about all kinds of things that get in the way of expressing what they're trying to do.

If you're a mere mortal, two things drive you nuts: memory allocation and data typing. And everything from Teco to Java bogs you down in various kinds of arbitrary limits. Even my PalmPilot has some rather strange arbitrary limits. All because designers think they know better than the user what will be good for the user. The only designer that good is God.

There are a lot of other languages in the world. Dennis Ritchie has said, "people need to understand that C and Pascal are really the same language." Perhaps Perl and C are the same by Ritchie's criterion, nonetheless, Perl does things differently. Among other things, it takes care of much of

Camel Lot IN THE Bilingual Unix

the business of memory allocation and data typing.

Especially when you're dealing with text processing, the atomic chunk of data is a bunch of text. It is not a tuple, a date, or a 13-bit floating-point number. It is just a thing, and Perl is conducive to dealing with chunks the way a human thinks instead of bending over backward to make things easy for the computer. We need to optimize both computer time and human time, but when push comes to shove, computer time is getting cheaper, and human time is (we hope) getting more valuable. The person has to come first.

Computer scientists like to throw around the term "orthogonality," and that's what they mean when they talk about minimal languages. They visualize a problem space as having a certain number of dimensions, and they want to be able to give you coordinates to say go east 15 yards, and north 25 yards, for example. They think that somehow or other that is making things easier for the computer and the human, but I don't buy that. People who hype orthogonality should be sentenced to draw everything with an Etch-a-Sketch.

If you're in a park and you want to visit the restroom you don't go due east 15 yards and 25 yards north. You make a beeline straight for the restroom. Humans love to take shortcuts; they love to diagonalize.

At the University of California at Irvine they didn't put in any sidewalks when they built the campus; they just planted grass. The next year, they looked for trails in the grass and put in sidewalks where they knew people wanted to go. And those sidewalks did not go at right angles all of the time. What I'm trying to do with Perl is formalize the shortcuts people want to take.

Right from the start Perl was designed to evolve. One of the reasons for the funny characters on the front of variables' names is because it protected the name spaces of the variables from reserved words and I could add reserved words and not break anybody's script. This sort of thing was intentional. Beyond that, I wanted to encourage input from the community.

That's not stating it strongly enough. A language without a culture is dead. A sense of participation is important for any open-source project, but is utterly crucial for a language. I didn't want people to merely say, "I know how to program in Perl." I wanted people to say, "I am a Perl programmer." When people achieve such cultural identity, many things suddenly become easy. In particular, a kind of self-organizing criticality takes place, and the proper number of leaders (and followers) seems to appear as if by magic.

Being the founder of the culture, I've naturally tried to impose a few of my own ideas. It's "officially okay" for people to speak subsets of the language. (It's even more okay for neophytes to speak a subset.) It's okay to establish subcultures, lingos, and fiefdoms within Perl culture. Anyone who wants to be can be a leader of something. If two people want to fight over the leadership of the same thing, we'll just call them different things and let Darwin take over from there. We compete by cooperating better. Hence, diversity is strength, because it lets different parts of the community intersect with other communities and Perl becomes a better "glue" language. C

LARRY WALL (lwall@oreilly.com) is a senior software developer with O'Reilly & Associates.