Pascal From the Quality Perspective
Bill Catambay, Pascal Central Editor

Pascal From the Quality Perspective

5-12-00

Editorial Note by Bill Catambay

Back when Apple began switching from Pascal to C, the road for Pascal programmers on the Mac began to get much tougher (it was bad enough fighting the pro-C attacks on Pascal). In the early 90's, developer support at Apple began to give Pascal programmers the feeling of being second-class citizens. It would often require a massive effort on the part of the Pascal community to ensure that Pascal interface files be created for their new technology. Pascal programmers applying for jobs at Apple even began to hide the fact that they preferred Pascal. McCarthyism seemed to have reared its ugly head, and Pascal was one of the targets.

Despite the challenges put forth, the Pascal community persisted. Apple, today, seems to be doing better with producing Pascal interfaces, and as we move into the Mac OS X arena, there appears to be methods of writing native OS X code in Pascal. One of the first developers to share the methods for writing Pascal code for Mac OS X happens to be a Quality Engineer from Apple Computer. Ron Drake, a proud and outspoken proponent of Pascal, agreed to do an interview regarding Pascal, sharing his perspective from the quality engineering side.

The Interview

Date of Interview:	4-10-00
Interviewee:	Ron Drake, Quality Engineer, Apple Computer
E-mail:	rdrake@apple.com
Subject:	Pascal and Apple

Q: How long have you been a programmer in Silicon Valley?

I learned Pascal at Bell Northern Research in 1981. The training staff there had a novel way of letting you choose which language you wanted to learn; you were shown samples and allowed to pick the one you liked best. No contest.

Q: What kind of machines and operating systems have you programmed on in the last two decades?

We had PDP 11-70's running UNIX at Bell Northern, so that's what I learned on. Hewlett-Packard had an especially useful flavor of Pascal running on its 3000 series machines. I programmed for a company in Palo Alto that did job automation software for the 3000 entirely in Pascal. That's where I learned the value of modularity, good formatting habits, and code reviews. I also did my own utilities on the IBM PC with Turbo Pascal--still the pound-for-pound champion of Pascal compilers. The last thirteen years have been spent at Apple doing a variety of testing tasks with Pascal. The latest was for the release of AppleShare IP 6.0 (ASIP). I wrote a tool that confirmed the function of the API's and glue code provided by Apple for server control calls. I also wrote a test tool in Delphi that exercises ASIP's support for Windows filesharing calls.

Q: Why do you think the push to move from Pascal to C in some companies is so strong?

It's getting harder to find Pascal programmers - good ones - every day. This is a function of what's being taught in the schools today. If you're teaching someone in elementary school a first language, anything other than C is seen as a waste of time and effort. I don't agree, of course.

This isn't to say that the trend can't or won't be reversed. Businesses are result-oriented. The convergence of the right tool with an argument that the cost of generating and maintaining a Pascal code base is inherently smaller than that for C/C++ will make a difference.

Q: When did Apple decide to switch to C? What do you think were the driving forces?

The switch was in full swing when I got to Apple, although there was still an abundance of support for Pascal. If I remember correctly, there was Think Pascal, MacPascal, and MPW Pascal. "Inside Macintosh" didn't start offering examples in C and Pascal until the early 90's, but the influx of university graduates beginning in the mid-80's had been introduced to C via UNIX. The power and reliability of the UNIX operating system made it--and C, by extension--the lingua franca of the bleeding edge.

There was also the attractiveness of braces in place of BEGIN-END pairs, the straightforward nature of ++ and -- to increment and decrement, the power and freedom of a pointer-based rather than a strongly typed language, and last--but certainly not least--the cachet of being one of those "in the know" on deciphering the resultant mess. That last point is important. C wasn't written to be readable or easily maintainable. It takes a lot of extra time and effort just to gain proficiency in reading it, much less learning how to produce efficient code; it's the nature of the language. That fact has cost the industry a lot of money.

Q: The developers of an OS have the best resources for building a compiler for their own OS, knowing all the in's and out's, etc.. Why do you think Apple dropped their efforts of building a compiler for the Mac OS? Do you see this philosophy changing in the future?

I'm constrained from commenting on Apple's future plans, but I can say that the movement away from MPW was a pure-and-simple business decision; Metrowerks was doing such a good job at providing the tools that they became the industry standard. I think they're to be commended for the crucial role they've played in helping Apple survive. I also think they should be commended for providing a Pascal compiler that will continue to serve our community well for some time to come.

I do believe, however, that the advent of MacOS X gives Pascal a chance for resurgence, not only on Macintosh, but on a variety of other platforms. Open source places the fate of the language into the hands of its adherents rather than with entities whose sole concern is the making of money. The foundation is already in place thanks to a couple of decent compilers available from the UNIX world. One important aspect of this is the return to console-based applications. The freedom from having to provide a graphical interface levels the linguistic playing field. An effective application coded in Pascal looks just like an effective application coded in C, the tools are universally available, and the barrier to entry posed by C's steeper learning curve is reduced.

There is still a thriving market for Delphi programmers on the Windows side because the product--Pascal-based though it may be--is still very reliable, powerful, and easy to maintain. The prospect of a true cross-platform solution--one that is closer to the initial "write once, run anywhere" goal of Java--is within the capabilities of Pascal. Pascal Central and the work of those who contribute to discussion groups and continue to insist on the availability of Pascal makes me confident that the language will grow and thrive.

I also would be delighted to see it return to its rightful place as a teaching language. Lord knows, we need it.

Q: How much Pascal programming are you allowed to do at Apple? How much of your Pascal efforts are done on your own time?

There's no programming involved in my work at the moment, although my background always helps.

As for my own time? I've already managed to compile a Carbonized Pascal application that runs well on both MacOS 9 and MacOS X. Although I suspect I'm not the first, I'm still proud to be one of the first. I'll continue to work with converting the sample code as time permits, but I'm also a screenwriter, so I have a limited amount of cycles to devote. I also want to be a good advocate for the language wherever and whenever I can.

Q: In your eyes, what do you see as Pascal's strengths over C and C++?

Foremost for me is clarity. The concept of a Pascal obfuscation contest borders on the ridiculous, yet the language sacrifices no power in comparison to C or C++, especially as it has been extended from the original.

Declaration of records/structures is clearer and more straightforward. I still find Pascal's FOR construct more accessible. The use of AND and OR instead of symbols obviate a translation that every English-reading human being HAS to make, whether it's made quickly or not. I needn't go further than a comparison of CASE vs. SWITCH.
String handling is inherently easier in Pascal than in C/C++.

One of the raps engineers had against Pascal was its strong typing. I've never wasted more time than when I tried to get a C compiler to use the type I wanted to use.

Casting...coercion...conversion...There has never been a construct in Pascal that I couldn't build if I needed it. And, when I've finished, the thing is what *I* say it is--not what the compiler thinks it is.

When I was programming HP3000's, I had a problem to solve; how to keep track of which records I'd used in a database. Thank goodness for the Pascal SET. What could be easier than "IF NOT (myRec IN mySet) THEN mySet := mySet + myRec"?!

Q: Do you believe, as other have asserted, that Pascal and C are basically the same language, but with a different syntax? What's your take on this?

Pascal and C may have some linguistic roots in common, but they are distinct languages with entirely different philosophies responsible for their creation. Pascal is a teaching language which, when extended, is also an extremely effective development tool. C is an extremely effective development tool that assumes that the programmer knows "the lay of the land." Pascal's strong typing allows a programmer to know certain things right from the start. C, and especially C++, require more "flight time" with the compiler in order to find out what is acceptable and what isn't. Some constructs that C compiles happily can be death at runtime. Some constructs that C warns you about at compile time are inconsequential at runtime... or not. Pascal presents the engineer with known quantities; C is a crap-shoot.

However, I can still tell when a C programmer has learned the trade via Pascal just by looking at the code. Good C reads almost like Pascal... almost.

Q: Some say Java takes C and improves its syntax, making it a more cross-platform, versatile and easy-to-read language, better than Pascal. Do you buy into that?

Java's C origins expose it to quite a bit of C's baggage, in my opinion. While it is more versatile and easy to read than C, it hides a lot from the engineer. This is great in small to medium-sized projects with a few engineers working on them. I've seen object-oriented code bases of any size become raging nightmares very quickly. One begins to lose sight of whether bugs are the fault of the programmer or the compiler. One begins to lose perspective on who created a data structure or even where it's declared and what it contains. This is a hallmark of object-oriented programming in general, but syntactic and other design shortcomings inherent in C just make this worse.

Q: Why do you think most Pascalians preferred to stick with Wirth's Pascal, and it's follow-on Extended Pascals, rather than follow him to Modula and his other compilers? Where do you stand in that regard?

My aim was to complete this interview without reference to "elegance" but I have failed. The original implementation by Niklaus Wirth is the foundation for a strong edifice in and of itself. I get the impression that there is nothing I can't do with a good Pascal compiler. All the tools are there; I just need enough imagination to use them. I think this is the attraction of the original and its extensions for most of us.

End of Interview

About Ron

Ron is a Bay Area native who grew up in East Menlo Park-East Palo Alto and is now living in San Jose. With a background in journalism, Ron has worked in the computer industry since 1980. His first programming experience was on an HP 9000 "calculator" at the U.S. Geological Survey writing a routine to compute rock densities. Ron is also a pianist-keyboardist (jazz, rock, rhythm

& blues) and a father. He enjoys old movies, Steely Dan, and Pascal.

Ron is not a spokesperson for Apple, and therefore his responses do not necessarily reflect the views of Apple.
