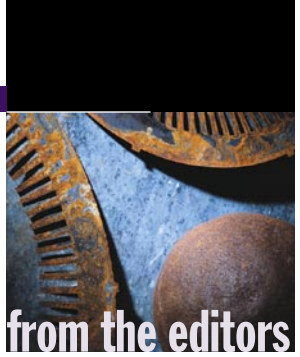

The Developer's Art Today: Aikido or Sumo?

David J. Brown, Queue Advisory Board Member

About once a month the *Queue* Advisory Board gets together for dinner to hammer out ideas for upcoming issues. Well, a few months back we fell into discussion about the problems surrounding software development these days. A few of us piped up straight away that tools are very important. Others countered, "Oh, sure, but do they help or do they hurt?" And so this issue was born: software development, tools, and whether or not they make us more productive.

When we returned to this proposed issue of *Queue* at a subsequent get-together to ask ourselves, "What's different these days?", the problem of scale in today's systems came up almost immediately: Many software systems are enormous, as is the number of folks one often finds working simultaneously on them. We wanted to ponder a bit about how on earth one gets a grip on all of it, given size alone. With this in mind, George Neville-Neil rendered the delightful phrase, "code spelunking," whereupon we immediately compelled him to expostulate: "Explain yourself, sir!" I'll come back to that …

As discussion continued, some of the skeptics in our midst revealed their hands. Such battle cries as "The only thing worse than tools is developers who trust them," and "Tools are not a substitute for (good programmers, good design, nor least of all) thinking," were heard.

A while later, we turned to a topic that I think deserves an issue or two in its own right: Given these large complex systems today, how the heck do you observe what they're doing? It's somewhat along the lines of debugging, but has a bit more of an "after you shipped it," runtime tracing, kind of spin to it.

Before the evening was over, a few of us who hadn't quite shouted ourselves out on the earlier tirade started croaking about the tools themselves. What are people actually using, and what are the problems with some of those things? We decided to conduct a survey to learn what the *Queue* readers are using.

Now you know this topic is pretty broad, and while I'd been reckless enough to produce an outline for its discussion—which somehow singled me out as the issue's prime volunteer—you can understand why I was a bit apprehensive to take it on alone. Enter soft-spoken Terry Coatta and the truly unsinkable George Neville-Neil, ready to coconspire. I must say that this issue owes greatly to their excellent work.

Do software development tools help or hurt productivity?

Together, here's what we came up with for the Development Tools issue:

We start off with Michael Donat's superb handling of one of the more acute sources of complexity these days, which comes from the recent vogue for multithreading right on up to the application level. What makes dealing with asynchronous behavior so hard, and how can you handle it?

Dear to the hearts of our "please don't check your brains at the door" crowd is Donn Seeley's excellent piece describing where software productivity really comes from. Read this. Reread this. We are not worthy!

Very neat, in my opinion, is the Phillips brothers' "No Source Code? No Problem!" How often have you found yourself with functionality that you simply had to keep going, but had only the binary for?

And finally, remember George's remark on spelunking? Well here's his explanation. The metaphor is just right, I think, and we "made him" write it down for you.

Along with these features, you'll find the results of the survey in which readers told us about some tricky problems they've run into, and the tools they like best.

There are also two other exciting pieces in this issue: Google is an epicenter of interest these days, and we were lucky enough to be able to interview Wayne Rosing—Google's vice president of engineering. We wanted to ask him how Google deals with these issues. This reminded some of us that grappling with huge tracts of code is far from all there is in today's developers' alligator-wrestling matches. There's another beast to contend with. Remember Niklaus Wirth's book, *Algorithms + Data Structures = Programs* (Prentice-Hall, 1978)? Yes, you got it: "Data" (OK, "data structures" to be precise). And—oh yes—indeed, they do have a little data to manage over at Google. Wayne and I also had a chance to speculate a bit on whether the meaning of "software development" and "tools" might have changed somewhat fundamentally,

given what's happened since the advent of the Web. I think you'll find his answers quite informative.

Finally, although not on the topic of developer tools, spam is more than just a concern, these days it's a problem crying out for a solution. Eric Allman is Sendmail's CTO. He deals with this a little. Don't you want to know what he learned about what Big Brother at the FTC is thinking about doing to "help out?"

Have fun. See you on the wires. Q

DAVID J. BROWN is the commodore of the Classic Yacht Association's Northern California Fleet. Once upon a time, he was a founder of Silicon Graphics and later received a Ph.D. at Cambridge University for describing a unified memory architecture for graphics workstations. After this, they gave him a job at Sun, anyway. Brown has worked on one or two big and horrible problems of software systems, including application binary compatibility in Solaris.

GEORGE V. NEVILLE-NEIL has worked with embedded systems for the past eight years as an integrator of final products and an implementor of off-the-shelf embedded operating systems. His work has centered on the networking aspects of embedded systems, but he has also done general work on broader aspects of the systems. Neville-Neil developed a device-driver model for networking devices used in VxWorks, worked on a multi-instance version of the Berkeley TCP/IP stack, and ported open source networking code to VxWorks. He is currently working on a new, commercial, dynamic host configuration protocol (DHCP) server at Nominum.

TERRY COATTA is currently the VP of development at Silicon Chalk, a small firm in Vancouver, BC, that is creating realtime collaborative software for use in higher education. Prior to that he was the director of development for distributed systems at Open Text Corporation, where he arrived via the acquisition of the Network Software Group, a consulting company of which he was the president. He has a Ph.D. in computer science from the University of British Columbia, where his area of research was distributed systems. He has worked with and continues to be interested in distributed component systems.