Editorial

What’s a Developer Supposed To Do?

Whil Hentzen

Well, if Visual FoxPro 5.0 isn’t out now, it’s either right around the corner or an earthquake recently leveled the entire left coast of the country. Doesn’t it feel like 3.0 just came out a few months ago? According to our surveys, a fair number of you have yet to migrate to Visual FoxPro, similar to the way that much of corporate America is still using some brand of Windows 3.1. But don’t think that you’ll be able to rest for long. The development team for Visual FoxPro will be on vacation for the weekend, then they’ll start work on the next version. Regardless of whether it’s called “6.0” or “Developer Studio FoxPro” or “SchmeeBase,” we have to be ready for it pretty soon.

Is your head spinning yet? It should be—and that’s just if you’re using FoxPro. Some of you are also developing in other languages: Visual Basic, Delphi, PowerBuilder, C++, and Java to name a few likely candidates. And each of these is going through frequent revision cycles as well.

How in the world are you going to keep up with all of these tools?

I could say something condescending like “just keep reading *FoxTalk* because that’s all you’ll ever need to stay cutting edge.” But you’ve heard that elsewhere, and you really wouldn’t believe it, would you? Another thing I could recommend is that you stick with FoxPro and stay away from all those other tools—who needs ’em, for example—and trying to do so sure complicates your life, right? Besides seeming self-serving, it doesn’t address the issue of incessant revisions of FoxPro itself, and thus I’m still searching for “the perfect answer” as well. So let me tell you what we’re doing at my shop, and see if you can find a nugget or two of wisdom in there somewhere.

First, let me explain that when I took over *FoxTalk* about 10 months ago, there were two of us in the shop, plus an occasional college intern doing odd jobs and the like. We now have something like eight folk: four or five full-time developers, a couple of support folk, and my assistant. So not only have I had to deal with the constant introductions of new tools, but I’m also working on getting and keeping everyone else up to speed as well. This double-edged sword puts additional pressure on me to find a solution to the ever-changing environment we’re working in.

Funny enough, our customers still expect high-quality applications, regardless of the tool we use to build them. Because of the higher and higher levels of pressure, it’s easier to let sub-standard code get shipped out to customers. One solution would to exhort my staff to “work smarter, not harder” or some other Dilbert-like mantra. But even if this approach would work, it would merely be a Band-Aid to a deeper problem.

The solution I’ve decided on is to focus on the process of software development—tools may come and tools may go, but the steps by which we create software remain the same. If the process is solid, then the degree of expertise with a specific tool is inconsequential as far as the customer is concerned. The process will prevent (or reduce the likelihood of) shipping junk to the customer. In other words, the customer is shielded from the inevitable mistakes made as you come up to speed on a new tool or a new version of a current tool. Of course, if you are a virtuoso in a particular language, this will still work. It’s just that the margin you can command will be greater.

We all know various pieces of the process, but hardly anyone has taken the time to spell out each step and document how each step is performed. This means, with each new project, the steps followed by the developer change, just as when cooking from memory instead of from a recipe. One time, the salt is added first, another time it’s added fourth (but twice as much is used), and yet another time it’s completely forgotten. And just as with cooking, the results you get will vary each time. Unfortunately, while uneven results with cooking may result in new and tasty concoctions, uneven results with software development are not desirable. As any quality guru will tell you, one of the cornerstones of quality is repeatability.

So we’re formally documenting our process, examining it, determining where we can measure the process and which of those measurements are of interest, and then performing the measurements. At the same time, we can use this documented process to train new developers. Once we have established a sufficient quantity of measurements, we can begin to determine how to improve the process, and thus improve the quality of our product. My goal for the next five years (through the year 2001—sort of eerie isn't it?) is for our shop to become reasonably proficient at the process of development.

In retrospect, this seems like a lot of work. Is it really worth it? In my (oh-so-humble) opinion, it definitely is. First, software is becoming more complex. By way of analogy, you can build a doghouse or panel your basement over the summer just by winging it. If, however, your job was to build a series of 25-story skyscrapers, and you had shorter and shorter time spans with which to work, your margin for error becomes much smaller and the repercussions of those errors become much more serious. You'll need proper tools, more knowledge, and a process to manage the construction.

And once you have that construction process down, you'll be able to adapt to new materials and tools and techniques a lot easier. You'll be able to finish the building on time, within budget, and with much greater certainty than if you were just guessing.

Second, it's a survival mechanism to handle the revision problem. Third, it's a way of differentiating yourself from the competition. And, finally, it's a route to a healthier bottom line. You'll get quality IS free in the long run because the time spent on a better process will result in fewer problems down the road, and that's money in the bank. ## I don't understand the "You'll get" edit in the previous sentence. The "You'll get" shouldn't be in there, I think, although the "because" does kinda sorta work. The sentence should be "it's a route to a healthier bottom line - quality IS free in the long run blah blah blah" does that make sense?

It's not my intent to write an epistle on the software development process in this article—we'll cover various topics in the future. I merely want to convince you to think about this as a possible avenue in your quest to deal with the frantic world we're living and working in. It was really unnerving to hear an executive from Sun Microsystems to use the term "perpetual beta" when referring to the set of products that Sun and their competitors are working on, but it underscores that the pace won't let up.

So that's our plan, and I'd like to suggest that you think about concentrating at least some of your efforts at doing the same.