Giving Birth

One of the most fun things I've done in my career is to help build the "1.0" version of Delphi. Okay, 2.0 was pretty fun too, but, as they say, there's nothing quite like the first time.

To be honest, when we started building Delphi 1.0, it was hard slogging. A lot of the tools out there, like PowerBuilder, SQL Windows, and Visual Basic, were pretty good. Long before we even had a prototype up and running, I was on the road talking to developers to understand the problems they were facing. To be honest, most of them were pretty happy. I remember particularly thinking that the VB corporate users by and large were just the happiest bunch of developers I'd ever met. Heck, they were even having fun!

I remember one particular meeting with about a dozen developers from a major airline. They told a not uncommon story of how a senior VP had decreed that they would use Visual Basic after he prototyped an application on a weekend. The developer told me that he didn't want to have any association with — *ugh* — Basic, but after trying it out for a while, he changed his mind.

Another time, I was meeting with a development team at a Wall Street foreign exchange. They showed me this tremendously impressive application for monitoring currencies — written in SQL Windows. It was beautiful. I thought, "Oh no, another satisfied customer. Time to move on." So after a demonstration I asked how they liked the application. His answer: "It's a dog." The response time was simply too long to even be considered for production use. After all, Wall Street practically defines the phrase "Time is money."

I saw these scenarios repeated in meeting after meeting, city after city. On the surface, customers seemed pleased with the productivity gains of "Rapid Application Development" tools. But as I delved further, I found the love affair often came to a bitter end when they tried to move from prototype to production. I found lots of spaghetti code out there, and DLLs written in C to make up for performance bottlenecks in applications written in PowerBuilder, SQL Windows, and VB.

It was some nine months into the two years of the development of Delphi 1.0 before we showed it to potential customers. There were two reasons for that. First of all, Delphi was an underground project that was truly secret. Heck, for the first year we had more code names than beta testers! Secondly, and perhaps more importantly, I wanted to make sure we understood customers problems rather than simply gauging a reaction to a demonstration. That way we could ensure we were building the right product, rather than fine tuning the wrong one.

When we finally started showing Delphi to customers, the reaction was dramatic. After all, we were solving the problems they had told us about. Our mantra: "Performance, Reuse, RAD, and Scalability." This helped us to not only define the product, but to communicate the benefits to the customer.

Oddly enough, one of the problems we faced was how to name the product. As Jerry Coffey pointed out in this column ["Symposium," *Delphi Informant*, January 1996], we were experimenting with all kinds of names. Although we had early on decided that "Pascal" should not be included in the name since it wasn't really meaningful except to long-time fans, we really hadn't made much progress on the name until a few months before its release.

Leading contenders included "Visual AppBuilder" (luckily, it was taken) "Application Architect" (too much like a CASE tool), "Client Builder" (sounds like a sales prospecting package), "Object Vision" (ahh, we used that already, didn't we?) and just about every combination of the words "Visual", "Power", "SQL", "Application", "Object", and "Builder" ("Visual Power SQL Application Object Builder" anyone?)

As we were in the late stages of selecting a name, whenever I'd do a presentation, whether to potential customers, sales reps, or third party vendors, I'd always ask them what they thought of the proposed names like "AppBuilder." Invariably, the response was lukewarm. Then they'd ask, "Why don't you just call it Delphi?" So in the end, we did.

Danny Thorpe, then on the QA team, now currently part of the R&D group, had came up with the original code name "Delphi" since it was to be a client/server tool connecting to the likes of Oracle among others. We had to come up with a code name for our first beta test and everyone felt that Delphi was acceptable. We had many later code names for internal use, external use, different countries, and at one point, I must admit, I randomly made up a new code name for every presentation, so that if there ever was a leak, we'd know from where it came.

Delphi 2.0 has come a long way since then. Our original goal was to address a few of the usability issues and also migrate to a 32-bit compiler and take advantage of platform features like OLE automation, OCXes, etc. Along the way, we introduced several major innovations like Data Module Objects and Visual Form Inheritance that increased code reuse.

The 32-bit compiler itself was actually started way back when the original 16-bit version of Delphi was started. At the time it seemed like a safe bet that "Chicago" would slip out of 1994 and that Windows 3.1 would still be a viable development platform for Delphi 1.0. We were able to have most of the VCL ported to 32-bits and running with the new compiler prior to the release of Delphi 1.0. So we were quite confident that the architecture would ensure compatibility with most code from Delphi 1.0, assuming it wasn't dependent on 16-bit data assembler, data structures, or unsupported API functions.

Although it's a bit too early to announce plans for the next version of Delphi, we're certainly working on a number of fronts to further reduce the amount of code folks need to write, and make it easier to support very large projects.

Zack Urlocker

Zack Urlocker is Director of Delphi Product Management at Borland International. The views expressed here are his own. He can be reached on CompuServe at 76217,1053.