Editors: Dave Thomas and Andy Hunt ■ The Pragmatic Programmers
dave@pragmaticprogrammer.com ■ andy@pragmaticprogrammer.com

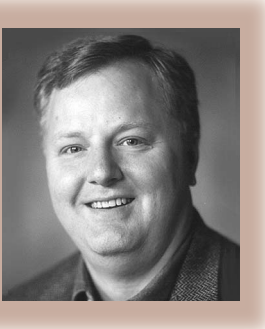
Imaginate

Andy Hunt and Dave Thomas

We receive email from our readers every day. Much of it relates their experiences in the field and how our writings apply. Every now and then we get a letter that really makes us stop and think.

I had posted a story on my blog about my young son's invention of the word *imaginate*:

> ***imaginate:*** *(v) to instantiate into reality by pure will of imagination*



A few weeks after I posted that, I received a letter from a fellow named Pete (I'll leave off his last name to give him a shred of privacy). Pete told me about a system he'd written that helped him rapidly create and deploy code for his clients and how it hinged on that word, "imaginate."

It seems he had this one client that had already suffered two failed attempts at this particular project. Each attempt was bid at over US$300,000 and estimated to take anywhere from three to four months. Between them, the company spent over $200,000 on services, $100,000 on legal fees because of the poor quality of those services, and more than a quarter-million on penalties because they still didn't have the software completed.

Pete looked at the project's requirements and thought they were pretty straightforward. So, he asked if could have a go at it. The client was necessarily skeptical, but she agreed to sit down with him and give it a try. Pete describes what happened next, using his home-brew application development system:

> *I "plugged in" one of the data structures that had been outlined in the RFP, pressed the "Go" button, and let the system do its work. A few seconds later, the client was entering some test data into the scheduling screen. Her comments: "This is simply amazing. Do you mean that I can imaginate like this whenever I think of something I want to try?" We both had a good laugh at her reaction. She went out and brought the whole department in to see the "system" she was designing "come to life" before her very eyes. The end result was that I got the assignment, it was completed in about three weeks, and they saved quite a bit of money.*

By now Pete had my attention. What a wonderful example of agile development! He had the ability to sit down with the user and create the software in real time. This is something we've always aimed for and looked to-

 0740-7459/04/$20.00 © 2004 IEEE

ward frameworks such as Naked Objects, new IDEs (integrated development environments), or new languages such as Ruby to help get us there.

So, I asked Pete about this marvelous technology he'd developed. What was it written in? Would he consider making it available as an open source project?

Imagine my surprise when Pete told me it was written in Cobol.

Worse yet, Pete chose Cobol for a very pragmatic reason: "not because of any particular bias for the language but for the additional features that come with the version I've used." He went on to say he'd written this system in various languages, dating all the way back to 1973!

Apparently, it doesn't require the latest technologies to make users happy.

It can't be done

But not all of his clients were as appreciative. In fact, a typical reaction he gets from IT departments to his rapid-development approach is "it can't be done." For 30 years, Pete's been in there pitching. The users and sponsors love it, but occasionally the IT folks say "it can't be done" and squash the project, throwing Pete out the door (hopefully that's just a figure of speech).

And of course they're right, aren't they? We, as an industry, love to build the grand frameworks that can solve all the world's problems in one unified package. With that in hand, you could in fact just sit down with the user and bang out a solid, robust application complete with security, navigation, user scripting, and so on, coupled with a repository of proven, debugged object prototypes that are customized as required.

But with all our resources, we haven't managed to do that very well yet. Applications that are thrown together quickly usually exact a steep price in the long run, leaving behind a quicksand-like pile of Visual Basic, Foxpro, or Perl code that isn't maintainable or extendable at all. And yet here's this guy who claims to be able to build software that lasts and delivers value, for as long as it's needed, and is simple and straightforward to understand, maintain, enhance, and extend.

How is that possible?

It can be done

Many of us get into the business of programming because we love the technology. We love listening to the crisp hum of the compiler as it builds a world according to our own whims. That's great motivation to get into the business, and it's probably a good angle if you're pursuing a research topic, but it doesn't cut it when it comes to programming in a corporate environment for business users.

Users don't care whether you use J2EE, Cobol, or a pair of magic rocks. They want their credit card authorization to process correctly and their inventory reports to print. You help them discover what they really need and jointly imagine a system.

Instead of getting carried away with the difficult race up the cutting edge of the latest technology, Pete concentrated on building a system that works for him and his clients. It's simple, perhaps almost primitive by our lofty standards. But it's easy to use, easy to understand, and fast to deploy. Pete's framework uses a mixture of technologies: some modeling, some code generation, some reusable components, and so on. He applies the fundamental pragmatic principle and uses *what works*, not what's merely new or fashionable.

We fail (as an industry) when we try to come up with the all-singing, all-dancing applications framework to end all applications frameworks. Maybe that's because there is no grand, unified theory waiting to emerge. One of the hallmarks of postmodernism (which some think is a distinguishing feature of our times) is that there's no "grand narrative," no overarching story to guide us. Instead, there are lots of little stories.

Pete's not trying to fix all the world's problems, just his.

Lessons learned

So what can we take away from all this? I think there are a few, very old-fashioned, very agile ideas in this story:

- *Users like results. They don't care about the technology.* Do you really care about the polycarbonate resins used to make your car engine? Or just that you get 80 miles to the gallon? What fab process was used to make the chip inside your cell phone?
- *Users like to be involved.* What's it like to be held hostage to a critical system that you depend on but into which you have no input? Try calling up your credit card company or long-distance provider and navigating their voice mail. Fun, isn't it? What would that have looked like if you'd been involved in its design?
- *Reuse is great, but use is better.* Pete solved recurring problems that presented themselves—not problems that might come up, but the ones that did come up. You don't need to solve all the world's problems; at least not at first.
- *Tools should support rapid development with feedback.* Our compilers, IDEs, and development tools need to support our ability to imaginate: to create what we want almost as fast as we can think it.

Unfortunately, our development environments are getting larger and larger, which makes development time—to say nothing of the learning curve—longer and longer.

If your tools don't support interactive design and rapid development with the end user or sponsor as an active participant, then you might as well scrap them and use Cobol.

Pete did.

Dave Thomas and Andy Hunt are partners in The Pragmatic Programmers and authors of the Jolt Productivity Award-winning *The Pragmatic Starter Kit* book series. Contact them via www.PragmaticProgrammer.com.