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

Verbing the Noun

Dave Thomas and Andy Hunt

Software development seems to be a discipline of artifacts; we developers spend our time producing *stuff* and attempting to find ways to measure just how well and how fast we make that stuff. And then managers tend to judge us by looking at the stuff—"that's a nice, meaty spec you've written, Dave."

Although this isn't surprising, it's a tad disappointing. Most of us joined the industry because we liked doing things. We get pleasure from the act of creation and from activities that surround the creation process. But as time goes on, we start to lose sight of this. Companies aren't interested in the process as much as the product. Managers can't measure the thought that goes in to a specification; they only see the document. In a previous company, Andy used to have a sign on his desk:

It is the artistry,
not the art.
It is the process,
not the product.
It is the journey,
not the destination.

The management made him remove it. And so we gradually learn to stop thinking about the doing and instead start concentrating on the end products.

Consequently, even though we're a bunch of folks who like to do things, we've become wedded to nouns, not verbs. Just look at the vocabulary of methodologies: requirements, design, quality, communication, tests, deliverables—all good solid nouns, and not a verb in sight. Yet increasingly, Andy and I are coming to believe that these things, these nouns, aren't really that useful. Instead, we see that the real value lies in the processes that lead to the artifacts' creation; the verbs are more valuable than the nouns. Let's look at just three of the methodology nouns (for now): requirements, quality, and deliverables.

The value of spending three months doing a requirements analysis is not the 150-page document produced. The report certainly doesn't capture the system's full nuance, and it will almost certainly become outdated as the project butts up against reality and the implementation adapts accordingly. No, the value of requirements is not the deliverable document, the artifact. Instead, the value is the process that everyone goes through to produce the document: the understanding gained, the relationships forged, the negotiations, the compromises, and all the other little interactions that share information.

So why do we insist on producing these doc-

 0740-7459/03/$17.00 © 2003 IEEE

uments? Why don't we all just interact for a while, hug, and move on to code? Well, it turns out that aiming to produce the document provides the framework in which the interactions can occur. The various sections of the boilerplate requirements document guide us to explore the different aspects of the business problem and to interact with the right set of people. The artifact's production gives us an excuse to perform all the really valuable activities. Methodologies that deemphasize the production of formal requirement documents find other ways of encouraging these interactions.

Quality is another important word. We plan it, measure it, hire folks to manage it, and buy big posters about it ("Consolidated Widgets Inc., where Quality is a Word on the Wall"). But again, you shouldn't use quality as a noun: you can't measure, draw, or describe it. All you can measure are its effects, the things that result from doing things a certain way. People mistakenly equate (say) bug rates with quality. But in reality, bug rates simply correlate with the way the software was written. If we do things a certain way, the bug rates will be lower and we can claim to have higher quality. Quality is part of the process; it's in the doing. Quality isn't a set of rules or a set of metrics; it's in the spirit of the smallest of our daily activities. Again, quality is a verb trapped in a noun's body.

Perhaps surprisingly, by considering the stuff we deliver to be nouns, we also encounter problems. In fact, some would argue that this is one of the biggest problems the industry faces. If we think of our software as an artifact, we view it as a kind of fixed entity, something that we ship before we move on to the project's next phase. But that's not the value of software, at least from our customer's perspective. Customers use software to address business needs, and those needs change constantly, either because the world changes or because our customers become more sophisticated in their understanding.

Whatever the reason, software delivery rarely represents the end of development, at least as far as the customer is concerned. Instead, the delivery simply represents a further refinement of our understanding of the requirement. Many developers treat a delivery as a chance to say, "Here's the software you asked for," but they should be saying, "How's this version?" This is why iterative development is so useful: it gives the business folks and developers numerous opportunities to refine their ideas and discover the system's hidden potential. If we view deliverables as nouns, we tend to ship them and run. But if we treat a deliverable as a verb—as something to do to help improve the way we add value to the business—we encourage communication, cooperation, and feedback between developers and customers.

Once we start thinking about this as a pattern, it can change the way we look at other artifacts we produce. Often, true value of some stuff doesn't lie in the stuff itself, but in the activity that created it. When we are heads-down, intent on producing an artifact by a particular deadline, it's worth stopping for a second. Is the artifact itself truly the valuable thing, or does the stuff that goes on during the artifact's production also have value? If so, are we extracting the maximum value from that production? Rushing from delivery to delivery, are we at risk of forgetting why we're here in the first place?

Let's end with a challenge. Think of some of the common nouns we fling around without a second thought (test, UML diagram, architecture, and colleague might be interesting starting places). Then, try to recast them (somehow) as verbs. Where do you find the value? Should we be emphasizing the doing of things more and the artifacts less? How?

Dave Thomas and **Andy Hunt** are partners in The Pragmatic Programmers. They feel that software consultants who can't program shouldn't be consulting, so they keep current by developing complex software systems for their clients. Contact them via www.pragmaticprogrammer.com.