Manifesto Elicits Cynicism

To the Editor:

The article titled “Agile Software Development: The Business of Innovation” (Jim Highsmith and Alistair Cockburn, Sept. 2001, pp. 120-122) is yet another attempt to undermine the discipline of software engineering.

According to this “manifesto,” we have come to value:

• Individuals and interactions over processes and tools. Translation: Talking to people instead of using a process gives us the freedom to do whatever we want.

• Working software over comprehensive documentation. Translation: We want to spend all our time coding. Remember, real programmers don’t write documentation.

• Customer collaboration over contract negotiation. Translation: Hagglng over the details is merely a distraction from the real work of coding. We’ll work out the details once we deliver something.

• Responding to change over following a plan. Translation: Following a plan implies we have to think about the problem and how we might actually solve it. Why would we want to do that when we could be coding?

Please excuse my cynicism. But I’ve been waiting a long time for software engineering to become a respected engineering discipline. While progress has been made, it has been painfully slow and frequently impeded by people within the software community.

In the software engineering profession, there are engineers and there are hackers. Hackers talk to people when they are stuck, since they often prefer to work without specifications. The number one priority for hackers is to produce something that “works” or at least appears to, which is easy if there are no written requirements. Hackers avoid planning. When they hit an obstacle, they do whatever is necessary to get something “working.”

It seems to me that this is nothing more than an attempt to legitimate hacker behavior. According to this manifesto, “Working code tells the developers and sponsors what they really have in front of them—as opposed to promises as to what they will have in front of them.” What this really means is that we can’t deliver what you want, so you’ll have to take what you get. What other industry can get away with this behavior?

The manifesto further states, “Processes, tools, documentation, and plans are useful. But when push comes to shove—and it usually does—something must give, and we need to be clear about what stays and what gives.” The issue here is that when push comes to shove, management caves into the pressure out of fear. Some hackers have management convinced that process, tools, documentation, and contracts—which are, by the way, the norm in every other industry—cause the pushing and shoving.

The software engineering profession will change for the better only when customers refuse to pay for software that doesn’t do what they contracted for. Management must recognize that successful businesses deliver what customers expect, in the agreed timeframe. Changing the culture from one that encourages the hacker mentality to one that is based on predictable software engineering practices will only help transform software engineering into a respected engineering discipline.

I hope I live long enough to see the day when this happens.

Steven R. Rakitin
Software Quality Consulting, Inc.
Upton, Mass.
info@swqual.com

MORE MARKUP REMARKS

To the Editor:

I read Neville Holmes’s comments about markup languages with amusement (“Crouching Error, Hidden Markup,” Sept. 2001, pp. 128, 126-127). As a long-standing LaTeX/TeX user, I experienced the same kind of shock when I first wrote a document in Microsoft Word back in 1995. The document was more than 100 pages long, which clearly crossed a line that still seems to exist in the latest version of Word.

Word’s WYSIWYG paradigm dates from the invention of the mechanical typewriter by Sholes and Glidden in 1867. However, today’s needs no longer make it possible to pursue the WYSIWYG goal completely, and Word clearly no longer offers WYSIWYG in every case.

TeX is often described as being old fashioned. It is very old indeed, but it certainly is not old fashioned because it implements the markup paradigm, which is very modern in that it introduces a layer of abstraction to the creation of documents. TeX is much like a programming language that automatically compiles an abstract source into a concrete executable image. The markup is the abstract source and the printable presentation is the concrete executable image. Automatically bridging such a gap increases productivity. Yet, many people who should easily be able to handle such a level of abstraction do not apply this principle to document production.

Continued on page 6
Letters

Continued from page 4

Somehow, the fact that markup introduces syntax in an explicit way frightens people. Indeed, the TeX syntax is not particularly evident, but new solutions such as XML/XSL strive to alleviate this. In an XML/XSL environment, a document writer uses XML, which has a trivial syntax that an XML editor can easily hide. Most of the complexity stems from the available vocabulary, which is defined by document type definitions. The advantage though is that DTDs can help to create complex applications. This offers many possibilities, especially in business, including standardization.

We need to think of document creation as entering structured information instead of as applying ink to paper. Automatic presentation generation should simply be taken for granted, especially in the corporate world.

Werner Donné
Beringen, Belgium
Werner.donne@re.be

Neville Holmes replies:

Other correspondents have recommended Scientific Word (http://www.mackichan.com), LyX (http://www.lyx.org), and WordPerfect (http://www.corel.com) as packages that successfully combine WYSIWYG with accessible markup, so I don’t think there is any basic incompatibility.

Although I deliberately avoided it, the reference to XML/XSL is interesting. But I do feel they made a basic mistake in not building their higher-level markup onto an underlying system that allows precise control of layout—whether for the screen or the printer—so that macrodefinitions can provide a good layout if the corporate user does in fact care about such matters. Then WYSIWYG or a similar easy-to-use front end can isolate end users from that scary markup.

To the Editor:

In his September column about markup languages, Neville Holmes fell into a common trap. Many computer literates think that markup is all that matters when it comes to typesetting. Holmes fails to understand the difference between a markup language and a typesetting language.
XML is a markup language; HTML is a typesetting language. TeX, in the form of the LaTeX or ConTeXt macropackages, might look like a markup language, but it isn’t. Typesetting is the art of creating the most beautiful layout given very specific constraints such as height, width, margins, and fonts. It requires value judgments a computer never will be able to make. For example, in reprinting his own works, whenever William Morris found a line that justified awkwardly, he altered the wording solely for the sake of making it look good in print (from Donald E. Knuth, *The TeXbook*, Addison-Wesley, Reading, Mass., 1984).

Regarding TeX and XML: Current macro packages like ConTeXt can read and typeset XML documents natively. It is here that we begin to understand that typesetting influences markup and to miss the ability to insert some leading here and there.

Neville Holmes replies:

When I worked with hot metal in a newspaper, the typesetters were the people who worked on Mergenthaler machines, and compositors put the lines of type into forms—their work was called composing. Earlier, I set type by hand into “sticks” that somebody else composed.

According to a correspondent in *Computer*’s October letters column, XML is a metalanguage. To me, both HTML and TeX are markup notations. HTML is very shonky as a markup notation. TeX is much better. LaTeX (and ConTeXt presumably, though this is a new one to me) is a package built on TeX’s macrofacility. In the background to my column was the thought that the people who introduced these new markup notations were taking giant strides backwards. They should have used a sound foundation (such as TeX) and built their higher-level notation using macrodefinition facilities. Then we would have had at least the possibility of well laid out documents, whether they were printed or displayed, as well as the bibliographic (misnamed “semantic”) notation.

On typesetting notation, I would have thought Knuth’s MetaFont to be such, rather than TeX.