In his May/June From the Editor column, Steve McConnell asked, How good is software construction? In a word, it stinks.

You would think it would be better given that so much has been researched and written. Unfortunately, most people haven’t progressed past their first year of work experience. They don’t read the literature. The same mistakes are being made in IT shops all over the place.

A recent boss of mine said he hadn’t opened an IT book since he left school 25 years ago. It showed. I moved on to a huge project that didn’t track schedules against deadlines nor adjust the project plan, and they didn’t review code. They were hugely over budget, and the quality was terrible.

After an impressive interview process for my current job, I was looking forward to it—until I got here. The code is terrible. They’re trying to test it into shape. The data schema doesn’t support the client’s needs, and they just try to work around it. The system is fragile, and things break every day. No one manages the team’s work or ensures product quality.

So, the state of IT is terrible, yes? Well, yes and no.

Professionally, the industry is terrible. Personally, it’s great. The fact that no one else is paying attention to the literature makes it easy to look good. Please don’t try to improve things.

James Cherryb
Software developer

How good is software construction? We spend a lot of industry air time (articles, books, conferences, consulting) on human-oriented soft things (requirements, use cases, design, arguments between CMM and the many agile methods) in the life cycle.

But we almost totally ignore the contribution of automated tools and processes to software productivity. If we had to do manually what tools (compilers, configuration management systems) and automated processes (scripts, software builds) do, that contribution would be more visible.

If the use of automation is considered a winning technique, why don’t we cut some of industry’s losses on the soft side and invest that effort in building more on the automated side? Businesses cut loser projects and fund winner projects. Sports teams cut losers and keep winners. People cut their losses and play to their personal strengths in their careers. Are we doing the same in the software industry, and playing to our strengths?
It seems not. We should spend more time on new, winning automated processes and less time patching up losing human processes.

Kevin Jameson
CEO, Codefast.com
jameson@codefast.com

Steve McConnell’s May/June column caught my attention—in particular, the question, How do you manage multiple releases? This has been the focus of my technical passion for the past 12 years.

I see fewer and fewer companies who build “just one.” I attribute this trend to customers who won’t settle for a generic solution that almost solves their problem, embedded systems that have to run on multiple variants of hardware, and increasing demands for integration of “everything to everything.” Even though most organizations create and support multiple flavors of their software, the industry still focuses on using technology and techniques for “one-of-a-kind” software engineering.

In the same way that configuration management provides abstractions and tools for managing software “variation in time,” we need abstractions and tools to manage software “variation in space”—that is, multiple system variations in a domain space. It is possible to divide and conquer the complex variation management problem (combining configuration management and customization management) into nine simple problems. Some of these problems are addressed by off-the-shelf CM technology, but others represent a void in current practices.

Think of this problem as software mass customization (the sidebar offers an annotated bibliography). Toyota can produce thousands of different Camrys from a single product facility. Their engineers don’t focus on the multiple Camry instances but on the single production line that produces them. In the same way, software organizations that build and maintain lots of product variations must shift their focus from individual products to software product lines based on UML and component-based engineering.
to the unified line that produces the different variations of their products. In the same way that manufacturing underwent a revolution from mass production to mass customization, software organizations that effectively embrace software mass customization will gain huge strategic and tactical advantage, thereby driving the whole industry in that direction. In Software Product Lines: Practices and Patterns, Paul Clements and Linda Northrop bring this point home in their wonderful sidebar entitled “E Pluribus Unum”—out of many, one.

Charles Krueger  
CEO, BigLever Software  
ckrueger@biglever.com

Steve McConnell asks, Why are we so gullible about silver bullets? Fredrick P. Brooks provides the answer in The Mythical Man-Month (Addison-Wesley, 1975): “All programmers are optimists…. ‘This time it will surely run,’ or ‘I just found the last bug.’” We can extend this to “This silver bullet will work.” It takes a lot of hard work to become cynical enough to overcome our natural optimism.

Brian Farquhar  
Sydney, Australia  
bdf@acm.org

They also claim that “there is rarely such a thing as too much documentation.” I have to disagree again. A well-written, clear, and elegant piece of code is much easier to understand than having to wade through thousands of pages of documentation. This is not to say that documentation is not needed, but, like anything else, let’s have some moderation.

Fred Rahmanian  
Senior consultant  
Source Tech Corp.  
fredr@acm.org

I do not necessarily disagree with the general point that James Whitaker and Steven Atkin try to make in their July/August article “Software Engineering Is Not Enough.” The authors discount the notion of program “simplicity, clarity, and elegance.” KISS is one of the most important aspects of good coding practice. Yes, there is nothing simple about a compiler, but the parts that make the compiler must be simple, clear, and elegant. Perhaps they forget that divide and conquer (one of the most important principles in all of computer science, in my opinion) can often result in the most simple, clear, and elegant solutions.

Also: Out-of-the-Box Thinking

Software Product Lines

The Business of the Software Product Line  
Hilfiker & Partners  
www.softwareproductlines.com

James J. Besemer  
President and chief engineer  
Cascade Systems Solutions  
jb@cascade-sys.com

It was our intention in writing this article to stir debate and discussion; we are pleased to see that we succeeded. It was also our intention to elevate the importance of developers and development skill, which we believe is the main cause for high-quality software and often the least appreciated task in software engineering. Whether one agrees with our specific examples is less important than the fact that discussion is taking place. We firmly stand behind our belief that software development is fundamentally a technical problem and that software processes must center around technology and discipline.
Letters

What’s “In It” for Me?

In response to the January/February 2002 From the Editor column “Closing the Gap,” I find that we use improved software processes as a defense mechanism in situations where we have to answer “no” to the following, deliberately loosely formulated, questions:

- Do you, as a software engineer, have time to carry out your work?
- Do you know how to “do it”?
- If not, are you confident that you will learn how to “do it”?
- Do you understand what is required?
- Can you work all right in your current office environment?
- Are you able to sneak some private technology research into your day?
- Does the job at hand challenge you in a positive way?
- Can you be optimistic about the project succeeding?
- Are you getting paid pretty well?

When I was working in a job where I could not say “yes” to these, I was aggressively trying to promote and introduce better software practices. Now, I have changed companies and can say “yes” to all the above. So, the appeal of (new) software engineering methods has dramatically faded.

I would even say that given the good situation I find myself in, any introduction of new process would have to be heavy on the side of “what’s in it for me,” or I just won’t be interested.

But isn’t the introduction of new practices a management decision, you ask? No. If I’m the guy who does the designing, testing, or implementing, then anything I don’t personally and full-heartedly buy into won’t make it. After all, I’m a software developer—I am prepared to quit (again) if my work conditions become stressful or unacceptable. Besides, it’s rarely management who comes and says we should spend more time on, say, the design process instead of coding.

I still have a positive approach: I am keen to embrace new practices as long as I’m convinced they’re any good.

Stephen Muires
Software engineer
Terma A/S
smu@terma.com