Thank you! Thank you! Thank you for confronting the “best practices” issue in such a direct and appropriate manner (Warren Harrison’s January/February From the Editor column “Best Practices, Who Says?”). Working in the software process improvement arena, I’ve been frustrated for years. I’m involved in some initiatives to collect and disseminate information about “best practices” (the name is not my choice) for software. Determining what practices to promote is proving to be a difficult task. If we try to adhere to a scientific approach, relying on solid data to support our decisions, we probably won’t publish anything.

My problem is even broader: I can’t seem to get a handle on what a practice is or isn’t. Is it different from, an element of, or broader than a process? An attribute of an engineering discipline? A step of a procedure? A quality attribute of an activity? Something we do? Or, any or all of these? I encounter the term being used to identify processes and engineering disciplines as well as specific elements of a procedure. For example, Configuration Management and Ensure Interoperability are sometimes referred to as key best practices. I’ve seen Make Sure You Have a Good Agenda identified as a best practice for integrated-product-team success, and Integrated Product and Process Development, a major segment of the CMMI, labeled a practice.

Every time I raise with colleagues the issue of needing to define terms, their response is, “We don’t need to spend time on that—we all kind of know what practices are.” Yet even within our small group, people use the term in many different ways. The range and scope of what gets labeled a practice is so broad that it seems to be totally unbounded and devoid of meaning. Where does practice fit among all these constructs: policy, method, approach, process, and procedure? What should the term’s scope be? Does it matter to anyone but me?

In a 2002 research project, in order to control the number of practices and capture the essence from many lists of practices, the researcher created metapactices—collections of similar or related practices. These included Configuration Management, Ensure Interoperability, and Technology Insertion. I’ve seen a similar pattern in the software community where a broader term is used to mean some set of related practices. The practices get abstracted to a practice area for communication purposes, and the practice area becomes the practice. If Configuration Management is a best practice, does that mean that whenever it’s implemented it is an instance of best practice? Is there bad CM, or is CM always good by definition?

I recently visited an organization that was having a conference on their best practices. In that context, they had labeled their current state of process (on a process improvement continuum) as their best practices. Using that logic, every organization is always implementing their best practices whenever they follow their defined processes.

The term “best practice” has lost its meaning. It takes on whatever meaning and scope an organization needs it to have at any given time. We need to replace it with terms you mentioned such as “effective,” “good,” or “value-added”...
practices. “Best” implies that some authoritative body has passed judgment. But who judges? What level of assessment is acceptable? Practices are seldom implemented in isolation, and most practices have interdependencies with other practices. The implementation context is often key to deriving value from a practice, yet the assessment process has not systematically addressed context. What’s best for organization A might not be good for organization B, but the practices that each employs might be equally valuable to their respective organizations.

In government and defense environments right now, there’s a big thrust to get organizations to become process focused by employing so-called best practices. This pressure is causing (in my opinion) organizations to label whatever they can a best practice in order to be able to report that they’re doing something. This causes degradation of the term’s meaning.

I’d like to get in touch with people who are as concerned about this as I am. Can you help me with some contacts or ideas? [See the related news brief on page 108.—Ed.]

(The comments presented here are mine alone; I’m speaking not as my company’s representative but as a member of the software development professional community.)

Ellen Walker
Analyst
ITT Industries—Advanced
Engineering and Sciences
ellen.walker@itt.com

Warren Harrison’s article on software development best practices was thought-provoking. I’ve discussed this subject with colleagues on several occasions and found many differing opinions. I completely agree with your move to restrict the term’s use in IEEE Software; the replacement terms you’re using seem far more appropriate.

Taking the term literally, referring to something as a best practice suggests it can’t be improved. This is misleading, as most practices evolve as development environments and tools evolve, and what’s a best practice today might not be so in the future. Adopting a so-called best practice might result in an organization not investigating or adopting other practices in the future because they believe that what they’re currently doing will remain a best practice indefinitely.

I believe we should continually monitor our software development practices and, when it becomes necessary, investigate possible alternatives that may better suit our particular situation and environment. Whatever practices we implement, if they’re effective in meeting the needs of our organization and its customers, they are appropriate.

Richard Harper
Software developer
Onyx Environmental Group
rharper@onyxgroup.co.uk

Continuous learning

Stephane Lussier’s article “New Tricks: How Open Source Changed the Way My Team Works” in the January/February 2004 issue was most informative. I liked his use of tragic irony in describing his team’s conversion from a “now the pros will show the hackers” point of view to a humble programmer’s “OSS programmers really know their stuff” point of view. Programmers (and other professionals) should never forget to keep learning from others’ successes or failures. The minute you think you know it all, you die intellectually.

My wake-up call came when browsing a bookstore’s computer section while on vacation in the US in the late eighties or early nineties. I was a systems administrator for BS2000 (a mainframe operating system generally unknown outside Europe). Most of the fashionable books didn’t apply to my field, so I was lucky to find Steve McConnell’s Code Complete, which I treasure to this day and always recommend to people in data processing. From his suggestion about personal improvement came my decision to join the IEEE and to subscribe to IEEE Software. It’s amazing how much value and knowledge I’ve gained from

WE POWER THE BEST BUSINESSES. AND THE BEST MINDS.

At SAP Labs, we design and implement innovative business software that helps millions around the world. And it’s people like you that make it all possible. That’s why SAP celebrates the individual – striving to let every idea and every voice be heard. And through the collaboration of these ideas, we’re empowering the best minds to help power the best businesses. In return for your efforts, we offer one of the most competitive benefits packages in the industry including three weeks paid vacation, premium health coverage, life insurance, flex hours, and even subsidized lunches. Visit www.saplabs.com to learn more and find out where you fit in.
acting on those recommendations and making that small investment.

Don’t be discouraged by people who say, “All this item contained was common sense.” They’re dead wrong as long as only a tiny percentage of IT professionals know that their job description should include much more than hacking out some quick lines of code.

Juergen L. Hemm
juergen.hemm@computer.org

The MDD Debate Goes On

Bob Glass’s March/April Loyal Opposition column “On Modeling and Discomfort” stimulates discussion of some pervasive, long-standing issues. The Model-Driven Development special issue (September/October 2003) describes, inadequately, a technique or approach that might be relevant and valuable to practitioners. However, neither academicians nor practitioners will know of the approach because of the gulf that has grown between them over the last 50 years.

I’ve been reading IEEE Software since the first issue. It’s improved over the years to the point where at least half of it is typically relevant to practitioners (thanks to Glass and others). Sometimes even academics’ contributions are applicable. In the MDD issue’s case, the writers didn’t use language or expository techniques that practitioners understand, so what looks like irrelevance might just be failed communication. Either way, practitioners aren’t getting much help from academicians.

Bill Dietrich
Arlington, Va.
bill@kastle.com

Good News, Bad News on Bugs

I read the March/April In the News story (“Whose Bug Is It Anyway?”) with interest. The world certainly has plenty of work to do on security bugs. One of the main problems is an overemphasis on fixing things after they break, including discussion about disclosure, “0days,” and patch management. Much better would be some hardcore discussion about how to build things properly.

The good news is that we now understand software is a major security problem. The bad news is that we focus only on easy bugs—defects we can locally identify with static analysis—and ignore more-difficult-to-find architectural flaws. Moreover, most application security vendors are concentrating their efforts on just one bug, the dreaded buffer overflow. There’s plenty of room for improvement. We should be discussing attack patterns, decompilation, rootkits, and more. Hopefully, we can change the discourse on security bugs for the better.

Gary McGraw
Chief technology officer, Cigital
gem@cigital.com

Inherited 1 MegaSLOC of MegaSLOP?

Our tools help developers understand, document, and maintain impossibly large or complex amounts of source code.

They parse Ada 83, Ada 95, FORTRAN 77, FORTRAN 90, FORTRAN 95, K&R C, ANSI C and C++, Java, JOVIAL, Pascal, & Assembly source code to reverse engineer, automatically document, calculate code metrics, and help your engineers understand, navigate, and maintain source code that has grown too large for one person (or even a group) to know.

Big projects aren’t a problem. The tools can parse and later manipulate very large amounts of code. 1,000,000 SLOC projects (and larger) are common among our customers.

We also focus on exceptional customer support based on rapid response from a real engineer, with bug fixes and new features incorporated into weekly builds.

Our tools:
- Ada 83 and Ada 95
- FORTRAN 77, FORTRAN 90, FORTRAN 95
- K&R C, ANSI C and C++
- Java, JOVIAL, Pascal, and Assembly
- Quick and easy use - no complicated or fuzzy setup. Immediately useful
- PERL/C/C++ API for custom reports and documentation
- Automatic creation of graphics and documentation
- Export to common graphics formats & Visio
- Cross-reference, references, and more
- Variety of hierarchical and graphical views (Including With Trees, Call Trees, Include Trees, Extended-By Trees, Ada Structure Graphs, and many others)
- Code colorizing for source editor and printing
- Rapid code navigation and editing
- Supported Languages:
  - Ada 83 and Ada 95
  - Java
  - ANSI C, K&R C, and C++
  - FORTRAN 77, 90, 95
  - JOVIAL
  - Pascal
  - Assembly
- Can create custom languages on request

Key Features:
- Fast on big projects
- Quick and easy to use - no complicated or fuzzy setup. Immediately useful
- PERL/C/C++ API for custom reports and documentation
- Automatic creation of graphics and documentation
- Export to common graphics formats & Visio
- Cross-reference, references, and more
- Variety of hierarchical and graphical views (Including With Trees, Call Trees, Include Trees, Extended-By Trees, Ada Structure Graphs, and many others)
- Code colorizing for source editor and printing
- Rapid code navigation and editing
- Supported Languages:
  - Ada 83 and Ada 95
  - Java
  - ANSI C, K&R C, and C++
  - FORTRAN 77, 90, 95
  - JOVIAL
  - Pascal
  - Assembly
- Can create custom languages on request

Download and try on your code:
http://www.scitools.com/

All downloads are fully functional, just time limited.

Scientific Toolworks, Inc.