

Earned-Value Management Terminology

To the Editor:

Although I enjoyed the article titled “Value-Based Software Engineering: A Case Study” (Barry Boehm and Li Guo Huang, March 2003, pp. 33-41), as a newly minted Project Management Professional and a contributor to the Project Management Institute’s Earned Value Analysis team, I was disappointed with the authors’ use of non-standard terminology.

The authors renamed the traditional terms BCWS (budgeted cost of work scheduled) and BCWP (budgeted cost of work performed) as EV scheduled and EV performed, respectively. They did not mention the third primary term, ACWP (actual cost of work performed), at all.

The earned-value management process is already confusing to many people. In an effort to help dispel this confusion, the 2000 edition of PMI’s body of knowledge replaced the terms BCWS with PV (planned value), BCWP with EV (earned value), and ACWP with AC (actual cost). Using this standard terminology will encourage clarity as project managers seek to establish discipline in software-intensive projects.

James T. Heires
Cedar Rapids, Iowa
james.heires@act.org

The authors respond:

We appreciate James Heires bringing up this issue. Lack of standard terminology can be and has been the



cause of many operational problems.

Unfortunately, what is nonstandard terminology in one organization is often the result of another organization’s attempt to provide standard terminology. We drew the terms BCWS and BCWP (and should have used ACWP instead of Cost) from another source of common project management and systems engineering terminology: *Communicating Project Management: The Integrated Vocabulary of Project Management and Systems Engineering* by Hal Mooz, Kevin Forsberg, and Howard Cotterman (John Wiley & Sons, 2003).

William Duncan’s foreword to this book provides an indication of the difficulty of getting common terminology adopted: “As the primary author of the 1994 and 1996 versions of the Project Management Institute’s *A Guide to the Project Management Body of Knowledge*, I believed that the document would support greater consistency. But though there are over 700,000 copies in circulation, even the Institute’s own publications frequently fail to use these terms as they are defined!”

We hope that the various stakeholder organizations, including the IEEE, can find a way to accelerate convergence toward the use of common terminology in this area. We’ll be ready and willing to adopt it.

PUTTING AGILE METHODS IN PERSPECTIVE

To the Editor:

I suspect that Guest Editors’ Introductions are not subject to the same review process as other papers published in *Computer*. Otherwise, how could the authors of “Agile Software Development: It’s about Feedback and Change” (Laurie Williams and Alistair Cockburn, June 2003, pp. 39-43) present a sweeping generalization such as the following:

“Software development cannot be considered a defined process because too much change occurs during the time that the team is developing the product. It is highly unlikely that any set of predefined steps will lead to a desirable, predictable outcome because requirements change, technology changes, people are added and taken off the team, and so on.”

Do the authors really believe that no software has ever been successfully produced by nonagile methods? It appears that “recent converts are the most passionate evangelists” applies here. I find no evidence of industrial software development experience on Laurie Williams’s Web page, and Alistair Cockburn’s Web site indicates that his experience is primarily as a consultant. Apparently, neither has worked in an organization with a strong software development process, as I have for 20 years.

Paul C. Jorgensen
Allendale, Mich.
jorgensp@gvsu.edu

The authors respond:

Most certainly, we do not believe that “no software has ever been successfully produced by nonagile methods.” In “Balancing Agility with Discipline” (*Computer*, June 2003, pp.

57-66), Barry Boehm and Richard Turner discuss when to turn to less agile methods. We do assert that successful software development processes are *empirical* as opposed to *defined*, whether they are agile or not.

An empirical process requires human observation and adaptation. A defined process runs by prescription without requiring intervention, such as can be programmed onto a robot. You can use defined approaches successfully if you employ empirical techniques (observe, inspect, adapt) whenever you encounter complexity. The failures occur when you ignore complexity and change.

Our industry's history supports this assertion:

- IBM's Santa Teresa Laboratory reported in the 1970s that requirements changed by 25 percent during the average software development project (Barry Boehm, *Software Engineering Economics*, Prentice-Hall, 1981). A company needs an empirical process to handle that amount of change.
- In "Iterative and Incremental Development: A Brief History" (*Computer*, June 2003, pp. 47-56), Craig Larman and Vic Basili report on the use of incremental and iterative (empirical) processes over four decades.
- Alistair Cockburn's research on organizations internationally over a 10-year period indicates that successful projects rely on perceptive intervention by alert staff, whatever process is used.

Our support of agile methods is based in part on our combined experience in industrial software development. Laurie Williams worked with defined processes as an industrial engineer for five years prior to joining the volatile world of software development at IBM. Alistair Cockburn's 28 years of project-lead experience includes commercial flight-simulator hardware design as well as software projects for

Norway's bank exchanges and various IBM clients. His research involves studying the differences between the way a group says they work and the way they actually work.

We stand by the sentences Paul Jorgensen quotes.



STANDARDS CLARIFICATION

To the Editor:

I enjoyed Craig Larman and Victor Basili's brief synopsis of the history of IID ("Iterative and Incremental Development: A Brief History," June 2003, pp. 47-56). However, I would like to comment on the authors' statement that the DoD "replaced Mil-Std-498 with another software acquisition standard, DoD 5000.2...." This is misleading. First, DoD 5000.2 is not a standard and it is not software-specific. Rather, it is a DoD instruction, titled "Operation of the Defense Acquisition System," that is meant to address all aspects of defense technology projects and acquisition programs, not just software.

Also, the information in DoD 5000.2 was not meant to "replace" Mil-Std-498. When the DoD instruction was issued and military standards were canceled, the underlying assumption was that industry should use appropriate commercial rather than military standards to guide system development. At the time, no comprehensive commercial standard for software development existed, so Mil-Std-498 was converted to EIA/IEEE Interim Standard J-STD-016, *Standard for Information Technology, Software Life Cycle Processes, Software Development Acquirer-Supplier Agreement*.

J-STD-016 conforms to the ISO/IEC 12207 international standard, *Information Technology—Software Life*

Cycle Processes. Although the normative guidance in the ISO/IEC 12207 is not as illuminating as J-STD-016 with respect to software development and maintenance, the ISO/IEC standard covers more territory than J-STD-016. Subsequent to the release of both J-STD-016 and ISO/IEC 12207, the IEEE/EIA published a set of three documents—IEEE/EIA 12207.0, 12207.1, and 12207.2—designed to provide clarifications, modifications, and guidelines for applying ISO/IEC 12207.

Rita Creel

Chantilly, Va.

r.creel@computer.org

BALANCING FEEDBACK AND FORESIGHT

To the Editor:

As Craig Larman and Victor Basili state, "Iterative and Incremental Development: A Brief History" presents a chronology of IDD, not an analysis. Nevertheless, an examination of a broader set of reasons for the waterfall method's continued presence helps in the promotion and enrichment of incremental and iterative methods.

Past iterations and increments may effectively constrain a system's future design and implementation. Without giving it much thought, engineers may use a single-pass process to assure that they are not painting themselves into a corner. In contrast, a smart iterative and incremental method calls for a high-level roadmap outlining the future direction for the system's design and implementation. Maintaining this roadmap requires a substantial investment, albeit more in expertise than in volume of work.

Second, in a broad class of cases, engineers must uncouple a target system's evolution from any concomitant changes in the system's environment. Again, using a single-pass process provides a defensive mechanism. If the only concern is the interface between the system's "ultimate" version and its environment, the singularity of the interface would, in theory, rule out any temporal compatibility issues.

A smart iterative and incremental method calls for an explicit architecture to maintain compatibility between an evolving system and its invariant or separately evolving environment. Defensive engineers may use an agile approach to adorn the single pass with iterations and increments as engineering needs arise. However, uncoupling a system's evolution from its environment is a *prima facie* requirement for an explicit architecture.

Third, in some cases, the customers are so totally occupied with present operational concerns they do not make time available to provide feedback on iterations and increments. In these cases, the waterfall method's illusion of getting things right the first time may be tempting. In contrast, a smart method may steer engineers to a solution like graceful fallback.

In the end, these three reasons are poor excuses for not applying an iterative and incremental method. But these methods must be about both feedback *and* foresight, in balance.

Rob Schaaf

Weston, Conn.

software.management@prodigy.net

DEFENDING FIREWALLS

To the Editor:

While I agree that using a firewall by itself as a primary means of achieving information security is an obsolete concept (William A. Arbaugh, "Firewalls: An Outdated Defense," June 2003, pp. 112-113), firewalls still serve a useful purpose.

Most current best practices consider a firewall to be an important component in a suite of controls and countermeasures that must be carefully coordinated through good policies and monitored with thorough audits.

Instead of comparing a firewall to deprecated technologies such as a moat and drawbridge, it is more appropriate to compare them to a locked door. Although a locked door probably won't prevent intrusion by a high-caliber thief, I would not opt to live in a house without a door. Likewise, I

would not eliminate the firewall on my computer system no matter how many loopholes there are.

Security is a fundamentally difficult problem from the business, political, and legal perspectives. Good technology is available, but a host of non-technical issues prevent most people from wielding these tools to their fullest potential.

Throwing more technology at this problem only hides or delays the inevitable consequences. A true revolution in information security will occur when technologists and managers begin working together to devise and enforce solid security practices that are undergirded by a supportive legal system.

Steven Hong

Washington, D.C.

shong@ubiqtec.net

The author responds:

I agree completely that firewalls are not yet obsolete, good technology exists today, and a large number of our problems are nontechnical in nature. However, the primary intent of the article was to point out that future technology changes (unrelated to security) may likely render firewalls completely ineffective. As a result, the protection that we have relied on heavily for years will not exist, and information devices must be capable of protecting themselves. The technology (and process) that I believe we must develop concerns not only how to have these devices protect themselves, but how to manage these devices since security and management are tightly coupled.

ANALYZING XML DEVELOPMENT TRENDS

To the Editor:

In "XML Raises Concerns as It Gains Prominence" (Industry Trends, May 2003, pp. 14-16), Steven J. Vaughan-Nichols fails to address several issues critical to a full appreciation of outstanding problems in the implementation of XML and related technologies.

First, the author mentions only DTD as the source of "an XML document's metadata rules." Although it is still in use, DTD is not the most efficient way of defining an XML document's semantics, rules, and structure. The current trend in the industry is to use XML schemas approved by the World Wide Web Consortium in May 2001 (www.w3c.org).

Second, the XML file size definitely can increase the load on corporate servers. However, it is not the file size per se but the resources required to parse XML documents that jam the servers. Furthermore, the assertion that compressing the XML document will "make the data stream more manageable" is only marginally correct. Compression reduces the XML document transmission time over the network, but after the transmission, the XML file requires decompression prior to or during processing.

Third, an important XML adoption problem relates to standardized XML schemas used in vertical applications or industrial domains. Many industry leaders support standardization initiatives and participate in the design of accessible XML schema repositories. As the XML schema is not usually included in the XML document, the effort required to fetch the schema for parsing may cause a host of additional problems; in particular, it could decrease the XML processing speed.

Finally, it is surprising that the author fails to mention several widely used or emerging technologies such as XSL, XSLT, XPath, and XML Signature. This oversight makes it impossible to perform a full analysis of XML-based software development and future trends.

Mikhail S. Brikman

Salem, Mass.

mikhail.brikman@salemstate.edu

We welcome your letters. Send them to computer@computer.org. Letters are subject to editing for style, clarity, and length.