XP’s Parts Greater than the Whole

“Piloting XP on Four Mission-Critical Systems” by Jerry Drobka, David Noftz, and Rekha Raghu (Nov./Dec. 2004) was excellent, but unfortunately, it wasn’t about XP. The title is a bit misleading, as are the abstract and introduction. The authors selected some aspects of XP and created their own process model. This certainly is the best and, in most cases, the only feasible approach, because XP has been cooked from best practices from around the SE world. But it’s no longer XP as the agile purist is selling it.

We learn from the article that some of XP’s elements are helpful. You can regroup some elements with other best practices to serve as a process model for large systems. This is nothing new for those who read *Microsoft Secrets* by Michael Cusumano and Richard Selby (The Free Press, 1995); they pointed out a decade ago that Microsoft uses some of what are known today as agile principles (better, they invented them) for large development projects.

We should emphasize a few points for practitioners here. XP and other agile approaches summarize best practices without gluing them too strongly; only in rare cases should you take the entire set of XP elements as an assembly. You can combine selected elements with other best practices, thus achieving a tailored version that best matches your needs. For example, depending on what market you address and how certain your requirements are, a customer will be on board—or not. While it’s always nice when customers care, especially so that you can put some of the responsibility on their shoulders, sometimes they won’t be present and your process will have to cope with it. Same for teamwork, which is another key agile principle. While trivial to establish in collocated small teams, it’s hardly feasible in large mission-critical systems. But workarounds can be established, such as clear-cut interfaces and collaborative environments.

Christof Ebert
Alcatel
christof.ebert@alcatel.com

Jerry Drobka, David Noftz, and Rekha Raghu respond:

We don’t believe our article’s title and abstract are misleading because our purpose was to document how we made XP work in our particular application. Therefore, the title “Piloting XP on Four Mission-Critical Systems” accurately describes what we did—we started with “pure” XP and tailored it for a good fit. We did, in fact, follow most XP practices. Software processes, as defined in textbooks, provide us with a roadmap; they aren’t written in stone and shouldn’t be treated that way. If we don’t let people use the term “XP” to describe their tailored versions of that process, we deny them the common language and definitions that “pure” XP provides, which in turn discourages communication and learning. And isn’t that why we read magazines such as *IEEE Software*—to share our experiences and to learn from others?
Ant and InstallAnywhere: Better Together?

In the Nov./Dec. 2004 issue’s Open Source column “Ant: Automating the Process of Building Applications,” Nicolás Serrano and Ismael Ciordia provide an excellent discussion of what makes Apache Ant different from typical build tools. They assert that Ant is competitive with Zero G Software’s multiplatform installer-authoring solution, InstallAnywhere, for distributing Java applications and that Ant is the “main” Java deployment tool. While in some cases Ant can compete with InstallAnywhere, the authors’ assertion is misleading. Many developers benefit from using InstallAnywhere and Ant together.

We designed InstallAnywhere to exploit the natural synergies it shares with Ant: extensible, cross-platform, Java-based software development. While there might be some overlap in functionality, InstallAnywhere provides a single executable file that can bootstrap a Java virtual machine, an optional graphical installation wizard, media-spanning for large installations, and team-based componentized configuration authoring. It also extends Ant by integrating installers into an automated build process, and it provides a built-in Execute Ant Script action that allows users to integrate Ant scripts with installation and configuration. InstallAnywhere Enterprise Edition even includes Ant in the download. Based on customer feedback, when it comes to software delivery, Ant and InstallAnywhere are better together.

Thank you for the opportunity to clarify this important point. I always look forward to reading IEEE Software’s Open Source column.

Tim Miller
Vice president, product strategy and technology, Zero G Software
tim.miller@zerog.com

You can reduce (but not eliminate) spam by using a longer email address, especially if you’re on a popular ISP.

Nicolás Serrano and Ismael Ciordia respond:

We appreciate your comments and interest in the column. Space doesn’t permit us to fully evaluate all available products. It’s difficult to say which is the best for each purpose, but for distribution of source code, Ant is clearly the most-used option.

Spam on the Increase

I just read Robert Glass’s column “Viruses Are Beginning to Get to Me!” (Jan./Feb. 2005), and I have an anecdote to share. A few months back, in response to increasing spam and viruses, my company implemented software that supposedly can better detect spam. If it thinks something is spam, it preends the subject with “Spam:”, and I can set up Outlook to automatically delete it or otherwise move it out of my inbox. In addition, if I get spam that’s not marked as such, I can send the message to a special email address with the subject line “false negative” so that the system will know it was spam. I can also report “false positives” (items marked as spam that I want to receive). After a while, the system will block the spam so I don’t even get them. That’s how it’s supposed to work, anyway.

Although my “data” is completely unscientific, I could swear the amount of spam I receive, marked and unmarked, has increased significantly. I noticed it immediately after the system’s introduction. I didn’t track how much spam I received prior to the system’s implementation, but I’d say it averaged one or two per day at the most. (I felt quite lucky compared to what I was hearing everywhere about how spam represents 50 percent or more of the typical user’s email these days). Now I routinely get five or six a day. It could be that I’m just more aware of it now, but I don’t think that’s the case.

Don’t know if there’s a cause and effect going on here, but it makes me wonder (and chuckle, now and then).

Dan S. Van Duine
vanduid@netzero.com

I thought I’d send my two cents’ worth in response to Robert Glass’s editorial on viruses. A couple of years ago, I signed up with the “local” cable company (actually part of a large nationwide company) and obtained an email address from them. However, as I already had email accounts with two other ISPs, I intended to just use the cable service as a gateway to the Internet and my ISPs. So I never used that email address in any way—I didn’t give it out to anyone, never entered it in any Web form, and never sent email to or from it. But I did routinely check it for incoming messages because the cable company used it to announce policy changes. After about two weeks, spam started to arrive in that mailbox in large quantities. How was that possible? I concluded that the cable company had sold the address to spammers, a fairly serious charge. However, I’d read that many spammers simply generate lists of addresses automatically, trying all combinations for popular ISPs up to a given length. I didn’t believe this until I used the cable service’s option to create additional mailboxes. I created another email address much longer than the seven-letter one that was getting the spam. I waited and checked for incoming spam. After a month or so with no email there at all, I concluded that the article I’d read was right. Conclusion: you can reduce (but not eliminate) spam by using a longer email address,
especiallly if you’re on a popular ISP such as cox.net, aol.com, compuserve.com, earthlink.net, and so on. (A friend of mine was so happy to get a nice four-letter-long email address—name@compuserve.com—equal to his nickname; he’s not so happy now.)

Having a domain name, I often provide a semi-bogus email address @software-production.com when I register on Web sites or otherwise provide email addresses to entities who really shouldn’t be sending me much. Emails sent to those addresses do get through, and I can tell when spammers get those addresses. Now, almost all the spam going to my regular email accounts comes from the Web sites I’ve maintained, in the typical “mailto:” form. The more Googled, the more spam.

John Erbland
Emergency Software Production
letters@software-production.com

Creating Software with Software

I read with great interest “The Tools at Hand” (Tools of the Trade, Jan./Feb. 2005), and I agree with Diomidis Spinellis’ complaint about software tools’ state of the art. Actually, little software is used in software production. Most software development work is done in tools that are little more than word processors—you can write something and then read what you wrote. Even compilers have limited verification capability and are usually unable to validate the content you feed through them.

Lack of tools is actually an old complaint. R.M. Graham et al. noted in 1973 “… current evaluation techniques make little or no use of a computer. Most analysis is done by hand” (“A Software Design and Evaluation System,” Comm. ACM, Feb. 1973). Whatever the issue, be it exercising more, saving for retirement, or building software tools, when we find ourselves saying decade after decade “we should do this” and decade after decade we don’t, there are usually some compelling reasons why not.

In my opinion, the fundamental business model underlying software creation is flawed. Businesses consider the software developer’s job to be to build a system that’s then shipped to the customer. As obvious as this might be, I think it’s wrong. The developer’s real job is to acquire knowledge—to learn. It just so happens that when developers acquire knowledge, we store it in a place that executes. If the knowledge is incomplete or inconsistent, the fact that we’ve put it somewhere that executes isn’t an advantage at all.

A software tool’s job is to encapsulate and make manifest knowledge about the activity of creating software. The alternative to storing this knowledge in a software form is to store it in a brain (experience) or on paper (documentation). The brain form has the disadvantage of being volatile and the paper form the disadvantage that it’s passive.

So, given that the software developer’s job is to move knowledge from a brain or paper to an executable form, why don’t we store the knowledge of software creation in software? I think many reasons contribute. Some relate to the psychology of knowledge discovery and some to the way we think about systems. But the key factor is that our business model drives us to consider only the encapsulation of knowledge for the end system as the “real,” important work. We consider everything else, including developing tools, to be something else—something less important.

And a final consideration about software development. It’s not unusual to hear people make a correlation between the industrial and “information” revolutions. However, the industrial revolution didn’t occur when we built steam engines—it occurred when we used steam engines to build steam engines.

Phil Armour
Vice president of systems development
Corvus International
armour@corvusintl.com