To print this page, select "Print" from the File menu of your browser

Totally awesome software?

"Extreme programming" sounds like no more than a marketing-driven fad, but fans are convinced that its rules hold the key to better code.

By Sam Williams

May 29, 2002 | The bug was trivial, nothing more than a missing letter. In a normal document, spellcheck would have caught it easily. In a software program filled with dozens of dyslexia-inducing commands, pseudo-words such as "CallOutOriginal," "CallOutCopy" and "CallOutFormRequest," it lurked invisible, and dangerous, like a piece of broken glass on a linoleum floor.

As Eli Collins, a programmer with the New York-based software firm Union Square Internet Development, scoured the list of error messages, Tom Clarke, Collins' colleague and "pair programming" partner for the afternoon, made the discovery.

"I think I see it," said Clarke.

"Where?" asked Collins.

"Right here," Clarke said, pointing toward a line of source code on the screen. "Looks like you left out the second 'i' in 'original' on line 172."

Within seconds, Collins, the designated typist, had fixed the error. With a click of the return button, the program was running once again through a battery of internal tests written by the two-man coding team over the last two weeks. This time around, the offending red tinge signifying a failed test was gone. The program's status bar showed solid green. Zero failures. Their newest feature, a point-and-click editing command, had received the green light, literally, and Collins and Clarke were ready to move on.

"Ah yes," said Clarke, leaning back while Collins piloted the machine. "Yet another example where having a second set of eyes helps."

In an era of cheap, portable computers and universal connectivity, the image of two people working over the same keyboard seems about as quaint as an RCA vacuum tube. For a growing number of programmers, however, it's the latest thing: "pair programming," a cornerstone tactic in an emerging grass-roots software development methodology sweeping the industry.

The name used to describe that methodology varies. Some call it "agile programming." Most call it "extreme programming," or "XP" for short. Whatever the name, the methodology's tenets boil down to an intriguing mix of age-old developer wisdom, newfangled coding tactics and a sugary-sweet layer of marketing-speak. Depending upon on whom you talk to in the software industry, it's either the biggest breakthrough since object-oriented programming or the biggest pile of hype since "push" technology.

"XP improves a software project in four essential ways: communication, simplicity, feedback, and courage," reads one XP booster site.

"XP is like a ring of poisonous snakes, daisy-chained together," counters another. "All it takes is for one of them to wriggle loose, and you've got a very angry, poisonous snake heading your way. "

The truth about XP, which is no relation to the Microsoft operating system Windows XP, actually lies somewhere in the middle. Communication is certainly the key component of any XP project. Designed to overcome the endemic problem of programmers promising one thing and delivering something totally different, the XP methodology is built around putting the face-to-face interaction between developers and customers -- not to mention developers and developers -- over the keyboard instead of over the conference table. XP's underlying article of faith is that if programmers and customers just communicate better, quality software will be the natural result.

Extreme programming also means giving up a little control -- acknowledging, at the outset, that software development is an imperfect process.

"It's the idea of admitting on Day 1 of a software project that you have no idea what your project's going to look like on a given future date and saying, 'I'm OK with that,'" says Colin Strasser, managing principal of Union Square Internet Development.

The alternative approach, as Strasser and other XP boosters are quick to point out, is the now familiar tale of software industry woe: Company A hires Company B to develop a multimillion dollar software program. The companies draft a contract, complete with specifications, and part ways while Company B writes the software. Eighteen months later, when Company B finally delivers, the resulting software looks absolutely nothing like the program specified in the contract. Or, even worse, Company B meets all the specifications but delivers a product riddled with bugs.

The history of the software industry is rife with high profile examples of this process. In 1997, the U.S. Internal Revenue Service scuttled a software upgrade to the tune of $3.5 billion. In 1995 the Denver Airport spent an extra $88 million, not including lost revenue, to debug its $193 million baggage handling system. Such examples, however, are less scandalous than the overall culture of shoddiness that seems to have infected the high tech industry over the last decade.

"There's a real tendency for business people and geeks to see development as a zero-sum game," says Kent Beck, the Oregon programmer now credited with coining the term "extreme programming." "The suits say, 'Give me more stuff, more stuff!' The geeks say, 'But I have to do a worse and worse job to give you more stuff.' I wanted to get away from that."

Beck traces the XP story back to early 1996. At the time, the Chrysler Corp. had called him in to help bring the company's new payroll system up to speed. In the process of putting the system through its paces, Beck made a troubling discovery: All the paycheck amounts were coming out false. Despite 18 months of development and millions of dollars, the software lived up to the original specifications but couldn't deliver correct data from one component to the next. After a few weeks, Beck found himself in the unwelcome position of telling Sue Unger, Chrysler's chief information officer, that she and her fellow executives had a giant lemon on their hands.

Instead of helping to overhaul the system, Beck offered his professional advice. He recommended that Unger dump the software and start over from scratch. To his lasting surprise, Unger accepted the advice -- with a Solomon-like twist. She put Beck in charge of the new project.

"I said, 'You don't understand. I'm a consultant. I don't do *in charge,'"* recalls Beck, laughing. "She said, 'You don't understand, I'm Sue Unger. You're in charge.'"

In the mad scramble to give Chrysler something it could use, Beck called upon many of the best practices he'd come across during his career. Early on, he decided to discard the usual whiteboard planning and view system design as an evolving, feedback-driven process.

"It was a combination of preparation and panic," Beck says. "I decided we were going to divide the project into stories. Each story would involve a specific feature that the customer wanted. It would also be something we could test, so we could show that we had fulfilled the customer request."

Writing tests before writing the code eliminated the need for exhaustive specifications, Beck says. It also eliminated the need for a large "quality assurance" team, which in most large-scale projects makes sure a software program obeys its specs.

Beck turned to pair-programming out of a similar desire for economy. The way he saw it, pair programming eliminated the need for extensive software documentation. If a programmer could communicate his ideas clearly to the programmer sitting over his shoulder, chances are, the programmer inspecting the code two years later would find it easier to understand. What's more, having a copilot on each machine would reduce the overall number of bugs.

The more Beck explained, the more solid the approach became. "By the 15th person, it had become fully streamlined," Beck says.

By the middle of 1996, Beck decided to give his management concept a name. Because other methodologies emphasized planning over programming, Beck decided that "programming" should be in the title. Still, "I needed an adjective that would be catchy, descriptive and defensible," Beck says.

Beck says decided to name his methodology "extreme programming" after noting the intensity of extreme athletes. "I wanted my teams to have that same sense of fearlessness in the face of challenge," he says.

Ron Jeffries, a friend and colleague called in to coach the Chrysler team, grabbed onto the term immediately. "The way it was described was we were taking all these practices that the best programmers already did and turning the dials all the way up to 10," Jeffries says.

Beck's strategy worked. By 1997, Chrysler had a new payroll system, and Beck and Jeffries were writing the first of many books on the XP "methodology" and convincing others to give it a try.

XP's ensuing rise from ad hoc method to next big thing is as much a testament to the evangelization skills of Beck, Jeffries *et al.* as it is to the virtues of the XP method itself. Five years after the fact, the number of XP testimonials trails well behind the number of XP books -- currently numbering more than a dozen on Amazon.com.

Such numbers prompt skeptics to wonder if "extreme programming" shouldn't be changed to "extreme hype." Doug Rosenberg, president of Itrix Software, a Santa Monica, Calif., company whose consulting and tutorial services now compete head-to-head with the growing number of XP tutorials in the marketplace, likens the XP phenomenon to a "brush fire" -- heavy on the flash but light on the fuel.

"Some of the ideas are actually pretty interesting," says Rosenberg. "They've actually contributed positively to the industry in terms of having people pay more attention to unit testing and automated testing frameworks. [But] the way they've marketed it, creating this whole fad phenomenon is objectionable to me."

Matt Stephens, another XP critic, prefers to zero in on the weaknesses of the XP approach itself. His August 2001 essay "A Case Against Extreme Programming" is the one that likens the XP approach to a ring of poisonous snakes. The ring analogy comes about, Stephens says, because of the XP proscription against "dabbling" -- adopting only certain elements of the XP approach. In order to make the XP methodology work, project managers must employ *all* the elements, according to XP proponents.

But Stephens considers certain elements risky. For example, XP places a low priority on source-code documentation, a common developer bugbear, on the rationale that well-written, well-tested code will be easier to understand by third party programmers than well-documented, poorly written code.

"If everything goes according to plan, XP might just get you there," Stephens warns. "If something goes wrong (e.g. your key programmer leaves; the on-site programmer, who is essentially the walking requirements spec, is hit by a bus; the janitor accidentally vacuums up your pile of story cards) then things can really go awry."

Beck downplays the argument that XP is an all-or-nothing approach. Still, he does note that the project leaders who report the most success with the methodology tend to be the ones who put aside the cultural obsession with rigid specs, an obsession inherited from less flexible fields such as civil and mechanical engineering, and approach XP's design-on-the-fly philosophy wholeheartedly.

"My strategy has been to say, 'Let's push it,'" Beck says. "If you think you're doing XP, but all you're really doing is a little bit of testing and weekly planning sessions, you haven't fundamentally changed the social contract yet."

After all, Beck notes, it was the gaping disconnect between management and software development that prompted him to develop XP in the first place. Rather than paint XP as yet another revolutionary trend in the software business, Beck prefers to describe it as a much-needed "return to civility." After nearly two decades of working on both sides of the line, Beck says he was looking for a way to eliminate, or at least diminish, the sociological disconnect that seems to characterize most software projects.

"Going into XP, I was very conscious that this had to be good for both sides," he says. "The suits can't say, 'We need these three things by Friday,' and the geeks can no longer say 'That's impossible.' With XP, you've got a situation where the programmers come back and say, 'We can give you two out of three. Which do you prefer?' Then the negotiating begins."

Whether or not other programmers and managers accept the bargain, XP is already lowering stress levels in some corners of the industry. Tom Clarke, the Union Square Internet Development programmer currently using techniques such as pair programming to fulfill his company's latest development contract, a Web site rebuild, says his first brush with XP was a very positive experience.

"I was working on a project for another company that involved sending out 50,000 e-mails every night," says Clarke, recalling his first brush with XP. "I'd go to bed wondering if the e-mails were going to the right addresses or if they were even going out at all. I had no way of knowing for sure. When I stumbled onto the notion of unit testing, I started experimenting with it. By the time I was done, I came up with a few tests that verified that, yes, the software did what I said it did. Suddenly, I found I was sleeping a lot better."

Since then, Clarke has helped to form a New York City XP interest group which meets in the Union Square Internet Development offices one night a week. As for his programming mate, Eli Collins, he reports a similarly positive take on the XP methodology.

"There's none of this worry about breaking things three days before release, which is a serious concern on most projects," he says. "[With XP], you can sit down with a client and if they say they want to do something new, you can say, 'OK. We're ready.'"

Recalling the earlier pair programming episode, Collins adds with a smile, "It's also a confidence booster when you know that stuff like a missing 'i' isn't going to come back and bite you."

- - - - - - - - - - - -

About the writer
Sam Williams is a freelance reporter who covers software and software development culture. He is also the author of "Free as in Freedom: Richard Stallman's Crusade For Free Software."

Sound Off
Send us a Letter to the Editor

Related stories
A unified theory of software evolution
Meir Lehman has been studying the life cycles of computer programs since he was a researcher at IBM 30 years ago. One of these days he's going to get it all figured out.
By Sam Williams
04/08/02

The joy of Perl
How Larry Wall invented a messy programming language -- and changed the face of the Web.
By Andrew Leonard
10/13/98

Disappearing into the code
A deadline brings programmers to the place of no shame. The body melts away, the mind races. Only one thing matters: Can you fix that demon bug? First of two excerpts from Ullman's "Close to the Machine."
By Ellen Ullman
10/09/97

GO TO: Salon.com >> Technology