Issue 11.09 - September 2003

The New X-Men

The Mountain Dew-fueled all-nighter is history. Today's supercoders work 40 hours a week. And two to a computer. It's called extreme programming - and it's revolutionizing the software world.

By Martha Baer

When Kevin Yu went to work for Hewlett-Packard in December 1999, he was a prematurely jaded 25-year-old programmer who had already lived through layoffs at Compaq and done a stint writing code at Digital Equipment Corp. Like most programmers he was withdrawn. He followed orders obediently. He felt removed. He worked alone.

Yu still codes at HP, but these days he's gregarious, passionate, and eager to solve problems of every kind. His attitude adjustment is the result of a new approach to writing software that's transformed practically every aspect of his job. The biggest change: Each afternoon, he pulls up a chair beside a fellow programmer, and the two of them share a single workstation - one monitor, one desk, one keyboard. It's common to find him in HP's Seattle offices shoulder to shoulder with teammate Asim Jalis, who stares into the screen as Yu "drives," both of them pondering aloud whether a new idea might work. Later, Jalis types as Yu watches, exclaiming periodically, "I get it!" when the spray of code makes sense. Sometimes one partner works the mouse while the other uses the keyboard, like a married couple finishing each other's sentences.

Michael Elins
"Mulder and Scully, Hewlett and Packard- pairing up is a natural way to work," says Asim Jalis, right, who programs in tandem with Kevin Yu at HP's Seattle office.

Yu is among thousands of coders who've discovered extreme programming, a method of software development that emphasizes constant feedback. Traditional coding devotes a huge amount of time to up-front planning, then demands rigid adherence to that plan. XP is different. Programmers spend relatively little time planning and instead dive into the writing, making course corrections as needed and allowing better ideas to emerge after snippets of code are tested and assessed. The result is a speedy loop: plan, code, test, release, plan, code, test.

At its core are 12 deceptively straightforward rules (see "Thou Shalt Work in Pairs," page 128), and Yu has embraced them all. He likes, for instance, the rigor of refactoring, revising code to make it as slim as possible. He appreciates the focus on quick releases, producing a one-step-better product several times a month. He relishes a work week capped at 40 hours. Mostly, though, what changed Kevin Yu is pair programming.

In a profession known for its lone wolves and silent cubes, in a culture routinely mocked for its social ineptitudes, putting coders up close to each other seems counterintuitive, even risky. Recent research into autism suggests that some software engineers may actually suffer from a genetic disorder that impedes their ability to interact. In *The Bug*, a new novel by programmer Ellen Ullman, the protagonist works under a tent rigged from a parachute and wears earmuffs as backup for his earplugs. Despite this, Yu is a convert to extreme programming. "XP," he says with a smile that forces his eyes closed, "is one of the luckiest things that ever happened in my career."

Jalis is also a believer: "Fred and Barney, Mulder and Scully, Gates and Allen, Hewlett and Packard - pairing up is a natural way to work."

The software development process is profoundly screwed up. According to the Standish Group, which conducts an annual industry-wide survey, 15 percent of all information technology projects get canceled outright, costing the sector $38 billion each year, and companies spend $17 billion annually on cost overruns. Those products that are finally released contain just 52 percent of the features customers asked for. Throughout the industry, projects are chronically late - only 18 percent hit deadline - and consistently, maddeningly flawed. Estimates of the number of bugs contained in shipped products run from one defect in every 1,000 lines of code to one in every 100. According to Watts Humphrey in his book *A Discipline for Software Engineering*, IBM at one time spent $250 million repairing and reinstalling fixes to 13,000 customer-reported flaws. That comes to a stunning $19,000 per defect.

Traditional efforts to improve matters have gone nowhere. Since the 1960s, IT managers have experimented with staffing and training, scheduling and tracking, team-building and quality control. Increasingly complex programming languages have touched off cycles of reinvention, first favoring rigid guidelines, then more organic ideals, then guidelines again - swinging, says Grady Booch, CTO of Rational Software and a student of development methodology, "from high ceremony to low."

It's dismal conditions like these that have given rise to extreme programming, with its promise of escape from historic failure. In the past few years, the XP message has circulated in books, mailing lists, conferences, user groups, and a gigantic Yahoo! community, which, since January 2000, has logged 76,000 messages. In London and Seattle, groups of techies get together on regular nights just "to pair."

Companies have started signing on. At places like IBM, Sabre, Symantec, even Domino's Pizza, teams of programmers and their managers have embraced XP. Hewlett-Packard has decided to turn a division of 80 into extreme programmers. What began less than a decade ago as a grassroots movement on the geeky fringe is now mainstream.

Now 29, Kevin Yu sits in a dingy, windowless room in HP's Seattle offices, folded up at various angles in his chair. Beside him is Asim Jalis, a 33-year-old Pakistani whose gentle voice and quiet movements would be prized in a nurse.

Across the cramped room sit two more coders at a corner desk. One is Lado Garakanidze, a massive, square-shaped

man whose low brow makes him seem perpetually depressed. He has the hands of a boxer; raised in the former Soviet republic of Georgia, Garakanidze spent his adolescence in the ring. At his right is Marshall Belew, a pale, clean-cut Texan who disappears at lunch to play ice hockey.

"Do you get it, Asim?" Yu says.

"No, but let's try it."

"The logic seems weird." They're sitting so close they're actually just muttering to each other.

"Yeah, kind of convoluted."

"It's the same idea as…"

"As what?"

"Hold it… Let me finish what I'm thinking."

Both pairs are fixated on their screens, like bird-watchers trying to detect leafy stirrings in the machine. If you saw them from the point of view of their monitors, they'd look like couples posing for party pics, heads together. Occasionally there are exclamations - "Ah!" and "Yesss!" - the kinds of sounds people make watching sports on TV.

"Oh no, do you see what I see?" says Belew.

Garakanidze slaps the table. "Ugh, I do."

They tip back in their springy chairs simultaneously. Throughout the building, these four guys seem like the only ones breathing. Elsewhere, the place is like an abandoned dotcom, or a natural history museum - silent humans at desks, motionless behind the glass.

"Now what do we do?" Belew asks, swiveling toward a slouching Garakanidze. Both grin. They're stuck, but they're enjoying the game.

Pair programming may be the most bizarre aspect of XP, but those who've tried it swear it's a blessing. With another brain at the ready, coders no longer suffer hours of frustration inside their own heads. While one partner types, the other can zoom in on misplaced semicolons or zoom out to look for flaws in the structure of the code. There's somebody there to celebrate with when things go right and someone to keep you on track when attention drifts toward email.

Michael Elins
Close-quarters programming takes longer, but studies show it's strikingly bug-free. HP's Marshall Belew, left, with his partner Lado Garakanidze.

By all accounts, the divorce rate is low. What's more, pairing is not a drain on productivity. Although concrete data is meager, a University of Utah study showed that paired programmers catch more defects than solo coders. Though the practice adds 15 percent per programmer to the time it takes to complete a task, the lost productivity is offset by having fewer bugs to fix. "Pre-XP," says Kevin Yu, "an eight-hour debugging session wasn't unusual. With XP we spend about half an hour." Meanwhile, 90 percent of the 41 coders in the study enjoyed working together more than alone, and almost all were "more confident in their solutions."

This self-assurance isn't bred only of collaboration. It's also the result of what is unanimously considered the method's most valuable principle: an emphasis on persistent, picayune, automated checks that determine in baby steps that a system is working.

"Test first" is an XP mantra. What it means is difficult for the nonprogrammer to grasp, but consider a simple metaphor: walking in the dark. The first stride on a pitch-black path is usually not a stride at all, but a tentative toe-tapping to make sure the footing is solid. In XP, an engineer can depart from the main body of his working program - feel with his toe off the path - to write a few lines that, when called, will confirm that the outcome he wants is the one his code will deliver.

Eventually, a team will amass hundreds, sometimes thousands, of these unit tests, which live permanently alongside the program itself, whether the system is moved to a new server, embellished, or upgraded. Thereafter, whenever the code is changed - a bug is repaired, a feature is added - engineers can immediately test the full system to make sure nothing got broken. Each time the tests are executed - and hundreds of them can run in seconds, one automatically called after another - a colored bar extends across the screen: green if the system passes every step of inspection, red if it fails. For extreme programmers, the satisfying green bar, which they may see dozens of times a day, is like the swish of a ball through the hoop or the right word locking into a crossword puzzle.

"OK, cross your fingers," says Garakanidze. He and Belew have just written about 10 lines of code.

Belew has the keyboard. "Are we doing this for foosball points?"

"OK, one foosball point." They stake out positions on whether the test will run without a glitch; the winner will be up a point when they head for the break room.

They watch a half-inch ribbon move across the screen. Lime green.

"Yes!" they both shout. It's impossible to tell who won the wager.

Kent Beck, who wrote the first book on extreme programming, is a computing rock star andprovocateur. Referred to by several colleagues as "the most brilliant programmer I know" and admired by coders all over the map, he's as open and eager as a kid. The bibliography of *Extreme Programming Explained*, his 2000 book that introduced the 12 rules, includes *The Structure of Scientific Revolutions*, *The Athletic Skier*, and *Sex Tips for Girls*.

But for all his impish enthusiasm, Beck's no sucker. He confesses that he wants to sell XP and sell it hard. Perhaps it's not the money he's after; his 20-acre ranch in rural Oregon where his five kids can roam, he explains, is wealth enough. But Beck is definitely waging a battle for hearts and minds - busily publishing, consulting, lecturing, and generally guaranteeing to liberate the troubled citizenry of programming.

"The marketing of XP is very deliberate and conscious," he volunteers. "Part of it is in co-opting the power of the media; I make sure I'm newsworthy from time to time. Part is in co-opting some of my publisher's ad budget." And part, he admits, is salesmanship: "If I'd had a charisma-ectomy in the beginning, XP would have gone nowhere."

Of course, you can't have that much attitude without making some enemies, and Beck has earned plenty of scoffs as well as some thoughtful critiques. "This industry has a long history of latching onto fads," says programmer and consultant Steve McConnell, "and then finding out, when people are less

hyped up, that these trends are not having nearly the success the evangelists promised." McConnell's charges against extreme programming are numerous. He argues that different projects call for different processes - you can't write avionics software for a defense contractor the same way you do an inventory system for a video store. Also, the rules are overly rigid - almost no one, he maintains, applies all 12 - and few are new; project management guru Tom Gilb came up with similar guidelines years ago. Given the sloppy premises and false advertising, McConnell concludes, "the phenomenon looks almost like mass hypnosis."

At the heart of the indictment leveled by detractors is a belief that XP is irresponsible. The relaxed approach to up-front planning and post-facto code documentation, the ceding of decision-making and accountability to customers, the pervasive hype - critics suggest that Beck's plan is little more than a justification for laziness.

Doug Rosenberg, who spent much of his career making software for the defense industry and now runs his own consulting outfit, is XP's most vitriolic opponent. He derides its followers as "extremos" and published a book devoted to *The Case Against XP*. His most damning attacks have been directed at the movement's iconic project: the Chrysler payroll system. It was the first use of extreme programming and is held up by XPers as a monument to the success of the approach. Rosenberg insists on being the voice of sobriety amid the intoxication: The Chrysler system, he reminds the masses, was canceled before it was completed and proved capable of less than a quarter of what was expected of it.

Beck has heard the charges and blames Chrysler for botching the project. Mostly, he dismisses complaints. "Extreme programming is an emotional experience," he maintains. "When you feel it, you understand." Then he adds a typically cheeky metaphor. "Talking about XP and trying it are two different things - like reading *The Joy of Sex* versus losing your virginity."

That Beck's eccentricities appeal to geeks is no surprise. What's peculiar is that his ideas are spreading into the heartland. George Azrak, senior VP of information systems at Domino's Pizza, remembers exactly how he became a believer in XP. In the fall of 2000, his boss requested a new sales-tracking system for the company, an interface for order takers at 7,000 locations that would handle the annual sale of 400 million pies. The backend had to track everything - toppings, crusts, sodas, sizes, delivery addresses - and, Azrak recalls, "We were supposed to have the first rev ready in nine months."

Sensitive and cheerful, with a Mr. Magoo face and powdery skin, Azrak had been producing software for three decades, and it was clear to him immediately: The job was too big, the time too short. "I knew that with traditional methods, I was doomed to fail." So he brought in XP consultants and hired several devotees of the method.

It worked. Azrak got his point-of-sale system, and he got it on time. The team relied most on the XP rule dubbed the Planning Game, a quaintly low tech approach to scheduling that includes "stories" written on old-fashioned index cards to describe a single feature a customer wants in the software. These scribbled tags replace the voluminous project descriptions of the past, typically written over the course of months (sometimes years) before work begins. An index card might, "Create a Cancel button that ends the current session," or "Add a Percent Change column to the report marketers download from the database" - the kind of task an engineer can code in a few hours. At Domino's, no story was so complex that it couldn't be realized in five days. By working in such small, manageable increments, programmers built the project organically, rather than being lashed to a design hatched at the outset. "There's something magical about XP," Azrak says now.

Tim Wise, who wears his thinning blond hair feathered back á la *Saturday Night Fever* and carries a 52-ounce Big Gulp into meetings, began doing extreme programming two years before he landed at Domino's. Ask him what's so great about pair programming and he'll give you an evocative analogy. "Say you're in Northern California, and you're driving those windy roads over the hills to Santa Cruz. It's night and it's rainy, and there's a brick wall on one side and a drop on the other. And say you don't know which exit you want, and you're trying to read a map while also watching for signs. Boy, wouldn't you be happy if there was someone beside you? Someone to study the map while you watch the road?"

Or ask Asim Jalis at HP, and he'll muse: "Once you've paired for a while, it's hard to program alone. I feel like my brain's half there."

Or ask a few XP programmers whether this new love of togetherness belies the old myth of the loner geek. Aren't coders supposed to be antisocial - comfortable only when enclosed in their abstract world, far from everything human? Is that image a fiction, made up by jealous outsiders?

No. No one disputes the reality of the awkward, math-minded introvert who doesn't belong. But they'll tell you that he's lonely.

"Thou Shalt Work In Pairs"

The 12 commandments of extreme programming

I _ The Planning Game
Meet with coders, managers, and the customer each week to schedule tasks for the next phase. Update the plan regularly.

II _ Small Releases
Put a simple system into production quickly, then release new versions on a short cycle.

III _ Metaphor
Create an analogy that expresses how the parts of the new system work.

IV _ Simple design
Design simply, and remove complexity at every stage.

V _ Testing
Write test programs that assure every portion of the code runs flawlessly before attempting a new task.

VI _ Refactoring
Edit the code to simplify, add flexibility, or remove redundancy.

VII _ Pair Programming
Write all code with two programmers at one machine.

VIII _ Collective Ownership
Permit anyone on the team to change code anywhere in the system at any time.

IX _ Continuous Integration
Bring components of the program together several times throughout each day to make sure they work in concert.

X _ 40-Hour Week
Strive to work no more than 40 hours a week. Never work overtime a second week in a row.

XI _ Onsite Customer
Include a real, live user on the team, available full-time to answer questions.

XII _ Coding Standards
Use agreed-upon styles and nomenclature to promote easy understanding of what the code does.

Writer at large Martha Baer (martha@wiredmag.com) wrote about deep code in Wired 11.02.