*By John V Levy, PhD, Certified Management Consultant*

**EXTREME PROGRAMMING PROVIDES A ROADMAP TO A CONSISTENT AND RELIABLE DEVELOPMENT PROCESS.**

# If extreme programming is good management, what were we doing before?

If you have managed a software project, you know what it's like to feel the push for results from your managers working against the pull of your dedication to the well-being of your development team. What if these two forces were not in conflict?

XP (extreme programming) comprises a set of principles that define a consistent and reliable development process. The developers of these principles did not discover them at random. Each of them corrects one or more deficiencies in previous development processes. This article outlines these principles.

## SMALL RELEASES

Users once required developers to implement major new functions at each release. When they keep up this practice, they never allow time for the code to stabilize and for programmers to see and fix subtle problems. Management always focuses on checking off the key deliverables on a status report to higher management levels. As a result, the steamroller of demand for functions always takes precedence over stability and robustness. In XP, the releases come frequently and with small but stable increments in function.

## SIMPLE DESIGN

Complexity appeals to people for various reasons. Designers like complexity because it tickles and challenges the mind. Software-development teams like it, once they create it, because it validates their sense of exclusivity to outsiders. And misguided mangers often like it because it increases the probability that their development organization will be large and long-lived, because complex designs require complex test tools, larger development teams, and constant efforts at improvement. XP emphasizes simple design, iteratively improved and refined.

## PLANNING GAME

Planning a project without involving the software designers and the business people together sets the designers against the specification and schedule, rather than getting their commitment to the spec and schedule. If the project leader alone does the planning, then the team feels that the leader has sold out or is part of management rather than part of the team. Finally, if the schedule is unrealistic, the team loses motivation to perform.

In the past, managers have consistently delivered the message to development teams that they are incompetent at estimating and controlling a project. Teams responded by meeting those low expectations. Intelligent people can learn to estimate effort levels when they have useful feedback and opportunities to practice. XP asks the team to estimate its work and improve its estimates at each iteration.

## WHOLE TEAM

Teams of competent people welcome leadership. But prima-donna technologists who lead projects tend to measure each person on the team by one criterion: Is this person smart enough to appreciate my brilliant design? Managers have promoted these prima donnas into project leadership and even into management without verifying their ability to lead. Real leadership inspires great performance by responding to each team member in the way that brings out the best in him or her.

First-level managers usually have to take on the role of keepers of the requirements. When requirements change, as they always do, fearful managers have often failed to communicate the changes to the development team, because they believe it will delay the next release or the whole project. They got away with this tactic because upper management rewarded first-level managers for fulfilling the original—often obsolete—project plan. This tactic was a triumph of good execution over good sense. In XP, customers and product managers are part of the team; they share responsibility for the scope of the project and can take advantage of emerging opportunities to develop the product as customer requirements change.

## TESTING

Separating the testing function from development puts test engineers in a lower status position. Separating quality assurance from development leads designers to believe that someone outside the design organization can ensure quality. Practitioners of XP develop test suites as the definition of project milestones by including customer-level functional tests in each test suite. This reaction is an effective countermeasure to impossible-to-meet or unclear milestones.

## COLLECTIVE OWNERSHIP

When managers promote and reward technology gurus who can't lead, they reduce buy-in by the other team members. This approach works against team commitment to the project goals. Current, accessible documentation is key to product update and maintenance, especially when team membership is changing and project requirements are evolving. Most designers resist documenting their designs because the task is larger than necessary. When the team believes in the whole process of design, delivery, and maintenance, then it will find ways to create the necessary documents.

Senior developers should not be the only ones to mentor junior developers, because such a practice reinforces prima-donna leadership, which ultimately undermines the team's effectiveness. The well-functioning XP team accepts ownership for the product life cycle and finds ways to engage every team member in the process, including training and development of junior team members.

## REFACTORING

Code seems fragile, so designers tend to freeze it as soon as they have evidence that it is working within spec. From this point of view, unchanging code is stable code. Modern software developers understand that, in the face of evolving customer requirements, evolving hardware platforms, and evolving network standards, frozen code rapidly becomes obsolete code. In XP, the countermeasure to change is having a clear understanding of the required functions and features, eyes-open awareness of the alternatives available for implementing them, and an ability to keep making trade-offs among performance, size, speed of development, and cost of development.

Refactoring in XP means constantly replacing code that works with new code that works better, works faster, or costs less to release. Once a team accepts this view, it can rapidly prototype without regard for performance, experiment with alternative implementations for speed, and improve memory requirements or network interactions. It ends up with much better products that cost less to maintain and to extend.

## PAIR PROGRAMMING

New team members need to learn in a work context. In XP, they grow quickly under the guidance of the whole team because there is at least one mentor for each member. Pair programming helps the junior member of the pair learn how the senior member thinks, and the senior member of the pair often learns just as much by having to explain his or her thinking. Senior-senior pairing is even better. It provides the beneficial effects of code review, design review, and peer pressure for maintaining good coding discipline. Team members, especially prima donnas, often resist pair programming, but more and more evidence is demonstrating that the practice is effective. Pair programming has the potential to produce better results than leaving the hard projects to individual "wizards" that many people thought were the key to success.

## CODING STANDARD

In old-style software management, managers left all decisions about coding style and documentation to the developers, because the managers had no idea how to specify what they wanted. Developers typically did not establish their own coding standards, because they respected their differences, and they often came from different backgrounds. Later, managers imposed coding standards based on something acquired from a vendor or from the quality-assurance department. These standards, superficially rational, caused resistance from the developers because they were not based on the developers' needs. Compliance came at the cost of creativity and a reduced designer commitment to code quality.

Coding standards are beneficial once the development team itself takes responsibility for the usefulness and effectiveness of the standard. Coding standards improve readability, portability, and quality when the team meticulously follows those standards. In XP, the team sees itself as responsible for enforcing the standards and for changing them when they get in the way of efficiency and effectiveness.

## SUSTAINABLE PACE

Vendors used to plan projects without the developers' participation. They used this top-down approach to estimating schedules and budgets without buy-in by the people who were going to do the work. Upper-level managers often insist on shortening project schedules without reducing requirements. As a result, a generation of software managers lived in a fantasy world where the real schedule had to be hidden while the function was frozen. The result was that no one trusted software-development schedules.

Those organizations depended on having software developers work 80 to 100 hours per week during the crunch to complete undermanned or overscheduled software projects. And the crunch was the standard state of existence. Working tired, the team made even more mistakes. This approach burned out the teams, burned out the managers, and destroyed credibility with everyone.

It takes a long time to create an environment in which top management believes software-schedule estimates. In XP, development teams create schedules and meet them in the presence of changing requirements. The manager's job is to defend the team's right to have a reasonable workweek. No one can sustain a 100-hour-per-week pace without losing efficiency and creativity. It is time for software teams to get a life and demonstrate that they can also consistently meet projected schedules. Management's job is to set up the conditions for this demonstration.

Designers used to assume that they could in a few hours or days build, integrate, and test a base level of software. This belief is unrealistic when few tools exist for automating the process. Integration and testing often require more than half of the project schedule, especially when no consistent module-level testing exists.

In XP, continuous integration and testing are antidotes to taking big schedule hits whenever a team builds a system. When the team knows that integration must occur every day or several times a day, then it will buy or build the tools to make the process easy to perform. When it is easy to detect which module is causing a failure during daily integration and testing, no big schedule surprises lurk at the next milestone.

Software management must evolve. Changing management thinking is the hard part of introducing effective development processes, such as XP. By studying what managers believed in the past, today's managers prepare themselves for this change in thinking. Enjoy the challenge.□

Author's biography

*John V Levy has been a management consultant since 1982. His practice focuses on product development with special attention to software development inside hardware-development organizations. He has held management positions with Quantum Corp, Apple Computer, Tandem Computers, and Digital Equipment Corp. He earned a doctorate in computer science from Stanford University (Stanford, CA) and holds bachelor's and master's degrees in engineering from Cornell University (Ithaca, NY) and California Institute of Technology (Pasadena), respectively. His Web site is at http://johnlevyconsulting.com, and you can reach him at 1-415 663-1818 or info@johnlevyconsulting.com.*

Acknowledgments

*Thanks to Kent Beck, Peter Christy, Tom Rolander, Jean-Dominique Savelli, Kevin Smith, and Mike Williams for their comments and suggestions.*