January, 2001

Editor's View

A New Way to Manage Development

Extreme Programming promises better code faster by challenging traditional development practices

One of the most widely discussed problems in our business is the number of application development projects that get bogged down and are either never delivered, are very late in delivery, or never work right. Close behind it is the problem of projects that are delivered to spec, but turn out not to solve the problem at hand.

There's a new programming methodology that aims headfirst at these problems. The technique is called "Extreme Programming" or XP for short. Its fundamental idea is to break a project into small, achievable steps with frequent releases and extremely rigorous testing. It's geared primarily toward high-risk projects or those where the requirements are either not yet well-defined or are likely to change frequently. XP is designed for small development teams, not projects with dozens or hundreds of developers.

The project is divided into a series of iterations (generally three weeks) and each iteration ends with a release. To determine what goes into each iteration, the first step is collecting the "user stories" for the project. The development team estimates the time to implement each user story, measured in "ideal development time," that is, if the problem was perfectly defined and there were no interruptions. User stories (which serve a similar role to use cases) are broken into smaller parts until they reach a size where the team estimates they can be completed in three weeks or less of ideal development time. The customer then indicates which user stories are most important, and those get priority.

One of the key features of XP is that you always do the simplest thing that could possibly work. Nothing is ever implemented just because "we'll need it later".

"Refactor mercilessly" is also one of XP's most fundamental rules. Refactoring means revisiting and restructuring code to simplify it. It's important not to get wedded to a piece of code simply because it's working. If you can rewrite it, pulling out redundancy and getting rid of unused functionality, the result will be easier to maintain.

Testing plays an extremely important role in XP. Unit tests (tests for one chunk of code, perhaps a class) are created either before or in conjunction with the code. The code can't be released to the project as a whole until it passes all the unit tests. Any time code is modified, it must be unit tested again and cannot be released until it passes every test. The customer helps to create acceptance (or functional) tests for each user story, as well. Both kinds of testing are automated, so that it's easy to test whether the particular module or the current version of the application passes.

As with any serious methodology, XP requires the use of coding standards. But with XP, standards are even more important than usual. That's because of perhaps the biggest difference between XP and traditional programming approaches, the way that people are managed.

Programmers work in teams of two (called "pair programming") rather than alone. Each pair sits in front of a single computer and works together, sharing the keyboard and mouse as needed. While this technique sounds terribly inefficient at first, it reminds me of some of the best problem-solving sessions. The proponents of XP claim that, in the long run, pairs produce the same amount of code as two people working alone, but the code is better.

Just as code is refactored on a regular basis, with XP, so are the pairs that produce it. XP calls for collective ownership of code, which means that rather than having one group always assigned to the user interface, while another owns the data, everyone works on everything at one time or another. Developers are moved from one task to another and pairs are broken up and reformed. In this way, everyone on the team is familiar with the entire project. If one area gets bogged down, others can work on it without the need for major training.

One of the rules for XP speaks to the part of me that believes in having a life. "No overtime" means what it says. The team works 40-hour weeks. With the opportunity to enjoy life, get enough sleep and refresh their minds, people are more productive.

There are other aspects to XP that I haven't mentioned here (see www.extremeprogramming.org and www.xprogramming.com for lots more information), but this is enough to give you an idea what it's about. XP has been used in a number of development projects, including some involving well-known companies.

XP is just starting to get discussed seriously in the VFP world. (Check out fox.wikis.com/wc.dll?Wiki~ExtremeProgramming~softwareEng.) Aside from the issue of asking people to change the way they work (which is a problem in any development environment), there are some apparent difficulties in applying XP to VFP development. The biggest is the automated testing requirement. Test harnesses have been created for a number of programming languages, but to date, there's none for FoxPro. I don't think this is an insurmountable problem, though. Another issue that's been raised is trying to apply iterative development and refactoring to databases, where small changes can wreak havoc with existing code.

I'm intrigued by what I've read about extreme programming. As an independent developer, I'm not in a position to put the whole methodology to work right now, but I'll be watching myself over the next few months to see which of its techniques I can use effectively.