

# extreme programming

Bruce Woods has over 15 years of software design and project management experience. When he's not managing nola's commercial development efforts, you can probably find him hunting fish with a bow and arrow or stalking deer and pigs near his home in Lacombe, Louisiana.

This past fall, while attending the JavaCon 2000 conference in Silicon Valley, I happened into a presentation given by one of the most energetic and dynamic speakers I have ever encountered. His name was Robert C. Martin, and, after intriguing the audience with several stories related to the formation of the universe he jumped at light speed into the main topic of discussion, eXtreme Programming, or XP. Each person in the audience was hanging onto Mr. Martin's every word. No one left before it was over. No other forum at the conference had as many attendees or elicited as much discussion.

As I listened, I recognized each of the principles and practices he described. At one time or another, I had successfully used these same practices—sometimes in desperation. Without knowing I was *XP-ing*, I had used pair programming to develop and debug complicated billing processes for Entergy. I had used a simple, minimal design to quickly commit verbal requirements to code. I had delivered much needed code in small incremental releases. I realized that I just might be a

natural born XPer.

A lightweight discipline of software development that emphasizes simplicity, communication, feedback, and courage, XP is designed for small teams that need to quickly develop software in an environment of rapidly changing requirements. XP focuses on the software produced rather than on rigid process and mountains of documentation. XP places a premium on analysis and design and makes meetings, reviews, and documents unnecessary.

Throughout an XP project, communication among programmers and between programmers and the customer is vital and built into the process, reducing what XPer's call the *project truck number*—the number of people on your project who can get hit by a truck without incapacitating the project. Much of the code developed on an XP project will end up being *throw-away* code as the customer reviews the products of each iteration and revises requirements accordingly.

I don't think any programmer alive today has worked on a project that has not been plagued by constantly chang-

ing requirements. XP maintains that the customer always has the right, at any phase of the development life cycle, to shift priorities, modify requirements—even to stop the project altogether. The beauty of XP is that even if customers decide to stop progress in the middle of a project, they will walk away with a functioning set of code.

The beauty of XP is the simplicity of design and requirements. Complex code is refactored until it is simple. The methodology is simple to follow as well. The chart at the top of the page shows an overview of the XP process.

User stories are handwritten, usually on forms developed by the project team. Each user story is accompanied by a functional test plan. During the *Planning Game*, the customer meets with the development team to estimate requirements. Programmers offer suggestions and insights on technical issues. Together, the development team and the customer plan iterations and agree to a release schedule. Such unknown quantities as architecture, feasibility, and third-party product evaluations are handled in *spikes* and

## extreme programming 12 key practices

1. The Planning Process—Requirements are simply stated on storycards, complete with test cases. The customer determines the business value of the features they desire, programmers develop cost estimates for desired features, and the customer determines which features should be developed and which should be deferred.
2. Small Releases—Early in the development life cycle, XP teams put a simple system into production and update it frequently. The customer is able to use the software and request adjustments as needed.
3. Metaphor—XP teams use a common system of names and descriptions.
4. Simple Design—XP teams use the simplest design possible to meet requirements. Unified Modeling Language diagrams may be used, you don't need a software product to create diagrams. Simple use case diagrams, class diagrams, and sequence diagrams are sufficient. First-rate design is ensured through the process of refactoring (see practice #6 below).
5. Testing—XP teams validate continually. Programmers write tests first, then develop software that fulfills test requirements. Customer-provided acceptance tests are also used to verify that needed features have been supplied.
6. Refactoring—Refactoring is analogous to a continuous software re-engineering cycle. Pair programming teams meet requirements with the simplest

assigned to a programming pair for resolution.

XP iterations are conducted in cycles lasting two to four weeks, depending on the size of the project and the development team. To start each cycle, the customer submits requirements and test plans and negotiates a release schedule. Programmer estimates are used as the basis for determining the deliverables of each iteration. If problems prevent any deliverables from being completed by the end of the iteration, the iteration does not continue. Instead, the product is reviewed with the customer in its current state. The customer may request delivery of the unfinished requirements in the next release or choose to eliminate them altogether.

The customer may also change or add requirements at each iteration. In any event, the customer is presented with working code at the conclusion of each iteration. The customer makes the call about what the development team works on next.

The customer, or an assigned tester, performs all functional testing. Bugs are reported to the development team and the customer may request that these are addressed at the next iteration. Upon customer approval, small incremental releases of the code are made available for production use.

If you'd like to learn more about XP, I recommend Kent Beck's book *eXtreme Programming Explained: Embrace Change*. I went through the book in two days and came away with a very good understanding of XP. For training and links to XP sites, [www.objectmentor.com](http://www.objectmentor.com) is a good place to start. </end>

code possible and don't worry about how to make it better. Duplicate code, complex code, and incomplete code are all refactored out. (The XP battle cry: Refactor mercilessly!)

7. Pair Programming—All production code is written by two programmers working at one machine. Software is produced at a consistent level and subjected to constant peer review. Junior staff mature quickly because they are paired with a seasoned project team member.
8. Collective Ownership—Code authorship is indistinguishable and the XP team shares code ownership.
9. Continuous Integration—During a typical XP day, teams build and integrate software multiple times. At the end of each day, the software is always left in a working state. Problem resolution is never postponed, so XP team members are always on the same page.
10. 40-Hour Week—Tired programmers make more mistakes. Overtime is seen as an XP execution failure.
11. On-Site Customer—Vital to any XP project is a customer representative who is familiar with all business aspects of the project, able to respond quickly to programmer questions, and empowered to adjust requirements and priorities.
12. Coding Standard—All programmers write code in the same way, resulting in code that is clear, consistent, and easy to maintain and enhance.