# Preface

This is a book about Extreme Programming (XP). XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements. This book is intended to help you decide if XP is for you.

To some folks, XP seems like just good common sense. So why the "extreme" in the name? XP takes commonsense principles and practices to extreme levels.

- If code reviews are good, we'll review code all the time (pair programming).
- If testing is good, everybody will test all the time (unit testing), even the customers (functional testing).
- If design is good, we'll make it part of everybody's daily business (refactoring).
- If simplicity is good, we'll always leave the system with the simplest design that supports its current functionality (the simplest thing that could possibly work).
- If architecture is important, everybody will work defining and refining the architecture all the time (metaphor).
- If integration testing is important, then we'll integrate and test several times a day (continuous integration).
- If short iterations are good, we'll make the iterations really, really short—seconds and minutes and hours, not weeks and months and years (the Planning Game).

When I first articulated XP, I had the mental image of knobs on a control board. Each knob was a practice that from experience I knew worked well. I would turn all the knobs up to 10 and see what happened. I was a little surprised to find that the whole package of practices was stable, predictable, and flexible.

XP makes two sets of promises.

- To programmers, XP promises that they will be able to work on things that really matter, every day. They won't have to face scary situations alone. They will be able to do everything in their power to make their system successful. They will make decisions that they can make best, and they won't make decisions they they aren't best qualified to make.
- To customers and managers, XP promises that they will get the most possible value out of every programming week. Every few weeks they will be able to see concrete progress on goals they care about. They will be able to change the direction of the project in the middle of development without incurring exorbitant costs.

In short, XP promises to reduce project risk, improve responsiveness to business changes, improve productivity throughout the life of a system, and add fun to building software in teams—all at the same time. Really. Quit laughing. Now you'll have to read the rest of the book to see if I'm crazy.

## This Book

This book talks about the thinking behind XP—its roots, philosophy, stories, myths. It is intended to help you make an informed decision about whether or not to use XP on your project. If you read this book and correctly decide *not* to use XP for your project, I will have met my goal just as much as if you correctly decide *to* use it. A second goal of this book is to help those of you already using XP to understand it better.

This isn't a book about precisely how to do Extreme Programming. You won't read lots of checklists here, or see many examples, or lots of programming stories. For that, you will have to go online, talk to some of the coaches mentioned here, wait for the topical, how-to books to follow, or just make up your own version.

The next stage of acceptance of XP is now in the hands of a group of people (you may be one) who are dissatisfied with software development as it is currently practiced. You want a better way to develop software, you want better relationships with your customers, you want happier, more stable, more productive programmers. In short, you are looking for big rewards, and you aren't afraid to try new ideas to get them. But if you are going to take a risk, you want to be convinced that you aren't just being stupid.

XP tells you to do things differently. Sometimes XP's advice is absolutely contrary to accepted wisdom. Right now I expect those choosing to use XP to require compelling reasons for doing things differently, but if the reasons are there, to go right ahead. I wrote this book to give you those reasons.

## What Is XP?

What is XP? XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software. It is distinguished from other methodologies by

- Its early, concrete, and continuing feedback from short cycles.
- Its incremental planning approach, which quickly comes up with an overall plan that is expected to evolve through the life of the project.
- Its ability to flexibly schedule the implementation of functionality, responding to changing business needs.
- Its reliance on automated tests written by programmers and customers to monitor the

progress of development, to allow the system to evolve, and to catch defects early.

- Its reliance on oral communication, tests, and source code to communicate system structure and intent.
- Its reliance on an evolutionary design process that lasts as long as the system lasts.
- Its reliance on the close collaboration of programmers with ordinary skills.
- Its reliance on practices that work with both the short-term instincts of programmers and the long-term interests of the project.

XP is a discipline of software development. It is a discipline because there are certain things that you have to do to be doing XP. You don't get to choose whether or not you will write tests—if you don't, you aren't extreme: end of discussion.

XP is designed to work with projects that can be built by teams of two to ten programmers, that aren't sharply constrained by the existing computing environment, and where a reasonable job of executing tests can be done in a fraction of a day.

XP frightens or angers some people who encounter it for the first time. However, none of the ideas in XP are new. Most are as old as programming. There is a sense in which XP is conservative—all its techniques have been proven over decades (for the implementation strategy) or centuries (for the management strategy).

The innovation of XP is

- Putting all these practices under one umbrella.
- Making sure they are practiced as thoroughly as possible.
- Making sure the practices support each other to the greatest possible degree.

## Enough

In *The Forest People* and *The Mountain People,* anthropologist Colin Turnbull paints contrasting pictures of two societies. In the mountains, resources were scarce and people were always on the brink of starvation. The culture they evolved was horrific. Mothers abandoned babies to roving packs of feral children as soon as they had any chance of survival. Violence, brutality, and betrayal were the order of the day.

In contrast, the forest had plenty of resources. A person had only to spend half an hour a day providing for their basic needs. The forest culture was the mirror image of the mountain culture. Adults shared in raising children, who were nurtured and loved until they were quite ready to care for themselves. If one person accidentally killed another (deliberate crime was unknown), they were exiled, but they only had to go a little ways into the forest, and only for a few months, and even then the other tribespeople brought them gifts of food.

XP is an experiment in answer to the question, "How would you program if you had enough time?" Now, you can't have extra time, because this is business after all, and we are certainly playing to win. But if you had enough time, you would write tests; you would restructure the system when you learned something; you would talk a lot with fellow programmers and with the customer.

Such a "mentality of sufficiency" is humane, unlike the relentless drudgery of impossible, imposed deadlines that drives so much talent out of the business of programming. The mentality of sufficiency is also good business. It creates its own efficiencies, just as the mentality of scarcity creates its own waste.

## Outline

The book is written as if you and I were creating a new software development discipline together. We start by examining our basic assumptions about software development. We then create the discipline itself. We conclude by examing the implications of what we have created—how it can be adopted, when it shouldn't be adopted, and what opportunities it creates for business.

The book is divided into three sections.

- The Problem—The chapters from "Risk: The Basic Problem" to "Back to Basics" set up the problem Extreme Programming is trying to solve and present criteria for evaluating the solution. This section will give you an idea of the overall worldview of Extreme Programming.

- The Solution—The chapters from "Quick Overview" to "Testing Strategy" turn the abstract ideas in the first section into the practices of a concrete methodology. This section will not tell you exactly how you can execute the practices, but rather talks about their general shape. The discussion of each practice relates it to the problems and principles introduced in the first section.

- Implementing XP—The chapters from "Adopting XP" to "XP at Work" describe a variety of topics around implementing XP—how to adopt it, what is expected from the various people in an extreme project, how XP looks to the business folks.

## Acknowledgments

I write in the first person here, not because these are my ideas, but rather because this is my perspective on these ideas. Most of the practices in XP are as old as programming.

Ward Cunningham is my immediate source for much of what you will read here. In many ways I have spent the last fifteen years just trying to explain to other people what he does naturally.

Thanks to Ron Jeffries for trying it, then making it much better. Thanks to Martin Fowler for explaining it in a nonthreatening way. Thanks to Erich Gamma for long talks while watching the swans in the Limmat, and for not letting me get away with sloppy thinking. And none of this would have happened if I hadn't watched my dad, Doug Beck, ply his programming craft all those years.

Thanks to the C3 team at Chrysler for following me up the hill, then storming past me on the way to the top. And special thanks to our managers Sue Unger and Ron Savage for the courage to give us the chance to try.

Thanks to Daedalos Consulting for supporting the writing of this book.

Champion reviewer honors go to Paul Chisholm for his copious, thoughtful, and often downright annoying comments. This book wouldn't be half of what it is without his feedback.

I have really enjoyed my interactions with all my reviewers. Well, at least I have gained tremendous help from them. I can't thank them enough for wading through my 1.0 prose, some of them in a foreign language. Thanks to (listed in the random order in which I read their reviews) Greg Hutchinson, Massimo Arnoldi, Dave Cleal, Sames Schuster, Don Wells, Joshua Kerievsky, Thorsten Dittmar, Moritz Becker, Daniel Gubler, Christoph Henrici, Thomas Zang, Dierk Koenig, Miroslav Novak, Rodney Ryan, Frank Westphal, Paul Trunz, Steve Hayes, Kevin Bradtke, Jeanine De Guzman, Tom Kubit, Falk Bruegmann, Hasko Heinecke, Peter Merel, Rob Mee, Pete McBreen, Thomas Ernst, Guido Haechler, Dieter Holz, Martin Knecht, Dierk König, Dirk Krampe, Patrick Lisser, Elisabeth Maier, Thomas Mancini, Alexio Moreno, Rolf Pfenninger, and Matthias Ressel.