Extreme Programming Explained

EMBRACE CHANGE

KENT BECK
WITH **CYNTHIA ANDRES**
Foreword by Erich Gamma

Second Edition

Praise for *Extreme Programming Explained, Second Edition*

"In this second edition of *Extreme Programming Explained*, Kent Beck organizes and presents five years' worth of experiences, growth, and change revolving around XP. If you are seriously interested in understanding how you and your team can start down the path of improvement with XP, you must read this book."

—**Francesco Cirillo,** Chief Executive Officer, XPLabs S.R.L.

"The first edition of this book told us what XP was—it changed the way many of us think about software development. This second edition takes it farther and gives us a lot more of the 'why' of XP, the motivations and the principles behind the practices. This is great stuff. Armed with the 'what' and the 'why,' we can now all set out to confidently work on the 'how': how to run our projects better, and how to get agile techniques adopted in our organizations."

—**Dave Thomas,** The Pragmatic Programmers LLC

"This book is dynamite! It was revolutionary when it first appeared a few years ago, and this new edition is equally profound. For those who insist on cookbook checklists, there's an excellent chapter on 'primary practices,' but I urge you to begin by truly contemplating the meaning of the opening sentence in the first chapter of Kent Beck's book: 'XP is about social change.' You should do whatever it takes to ensure that every IT professional and every IT manager—all the way up to the CIO—has a copy of *Extreme Programming Explained* on his or her desk."

—**Ed Yourdon,** author and consultant

"XP is a powerful set of concepts for simplifying the process of software design, development, and testing. It is about minimalism and incrementalism, which are especially useful principles when tackling complex problems that require a balance of creativity and discipline."

—**Michael A. Cusumano,** Professor, MIT Sloan School of Management, and author of *The Business of Software*

"*Extreme Programming Explained* is the work of a talented and passionate craftsman. Kent Beck has brought together a compelling collection of ideas about programming and management that deserves your full attention. My only beef is that our profession has gotten to a point where such commonsense ideas are labeled 'extreme.' . . ."

—Lou Mazzucchelli, Fellow, Cutter Business Technology Council

"If your organization is ready for a change in the way it develops software, there's the slow incremental approach, fixing things one by one, or the fast track, jumping feet first into Extreme Programming. Do not be frightened by the name, it is not that extreme at all. It is mostly good old recipes and common sense, nicely integrated together, getting rid of all the fat that has accumulated over the years."

—Philippe Kruchten, UBC, Vancouver, British Columbia

"Sometimes revolutionaries get left behind as the movement they started takes on a life of its own. In this book, Kent Beck shows that he remains ahead of the curve, leading XP to its next level. Incorporating five years of feedback, this book takes a fresh look at what it takes to develop better software in less time and for less money. There are no silver bullets here, just a set of practical principles that, when used wisely, can lead to dramatic improvements in software development productivity."

—Mary Poppendieck, author of *Lean Software Development: An Agile Toolkit*

"Kent Beck has revised his classic book based on five more years of applying and teaching XP. He shows how the path to XP is both easy and hard: It can be started with fewer practices, and yet it challenges teams to go farther than ever."

—William Wake, independent consultant

"With new insights, wisdom from experience and clearer explanations of the art of Extreme Programming, this edition of Beck's classic will help many realize the dream of outstanding software development."

—**Joshua Kerievsky,** author, *Refactoring to Patterns,* and Founder, Industrial Logic, Inc.

"XP has changed the way our industry thinks about software development. Its brilliant simplicity, focused execution, and insistence on fact-based planning over speculation have set a new standard for software delivery."

—**David Trowbridge,** Architect, Microsoft Corporation

Extreme Programming Explained

Second Edition

The XP Series

Kent Beck, Series Advisor

Extreme Programming, familiarly known as XP, is a discipline of the business of software development that focuses the whole team on common, reachable goals. Using the values and principles of XP, teams apply appropriate XP practices in their own context. XP practices are chosen for their encouragement of human creativity and their acceptance of human frailty. XP teams produce quality software at a sustainable pace.

One of the goals of XP is to bring accountability and transparency to software development, to run software development like any other business activity. Another goal is to achieve outstanding results—more effective and efficient development with far fewer defects than is currently expected. Finally, XP aims to achieve these goals by celebrating and serving the human needs of everyone touched by software development—sponsors, managers, testers, users, and programmers.

The XP series exists to explore the myriad variations in applying XP. While XP began as a methodology addressing small teams working on internal projects, teams worldwide have used XP for shrink-wrap, embedded, and large-scale projects as well. The books in the series describe how XP applies in these and other situations, addressing both technical and social concerns.

Change has come to software development. However, change can be seen as an opportunity, not a threat. With a plan for change, teams can harness this opportunity to their benefit. XP is one such plan for change.

Titles in the Series

Extreme Programming Applied: Playing to Win, Ken Auer and Roy Miller

Extreme Programming Explained, Second Edition: Embrace Change, Kent Beck with Cynthia Andres

Extreme Programming Explored, William C. Wake

Extreme Programming for Web Projects, Doug Wallace, Isobel Raggett, and Joel Aufgang

Extreme Programming Installed, Ron Jeffries, Ann Anderson, and Chet Hendrickson

Planning Extreme Programming, Kent Beck and Martin Fowler

Testing Extreme Programming, Lisa Crispin and Tip House

Extreme Programming Explained

Second Edition

Embrace Change

Kent Beck
with Cynthia Andres

Addison-Wesley
Boston

Publisher: John Wait
Editor in Chief: Don O'Hagan
Acquisitions Editor: Paul Petralia
Managing Editor: John Fuller
Project Editors: Julie Nahil and Kim Arney Mulcahy
Compositor: Kim Arney Mulcahy
Manufacturing Buyer: Carol Melville

Library of Congress Cataloging-in-Publication Data

Beck, Kent.
extreme programming explained: embrace change / Kent Beck with Cynthia Andres. — 2nd ed.
p. cm.
Includes bibliographical references and index.
ISBN 0-321-27865-8 (alk. paper)
1. Computer software—Development. 2. eXtreme programming. I. Title.
QA76.76.D47B434 2004
005.1—dc22

2004057463

ISBN 0-321-27865-8
Text printed in the United States on recycled paper at Courier in Stoughton, Massachusetts.

Text printed in the United States on recycled paper at Courier in Westford, Massachusetts.
Tenth printing, January 2012

To Cindee

Without you, this book would still be about programmers hiding in a corner. Without you, I would still be one of those programmers.

This page intentionally left blank

Note To Programmers

Even programmers can be whole people in the real world. XP is an opportunity to test yourself, to be yourself, to realize that maybe you've been fine all along and just hanging with the wrong crowd.

This page intentionally left blank

Contents

Foreword to the Second Edition

Wow—the second edition. I cannot believe that five years have already passed since the appearance of the first edition. When Kent pinged me to write a foreword to the second edition I asked him for a manuscript version with change bars. What a silly request—the book is a full rewrite! In the second edition of XP Explained Kent revisits XP and applies the XP paradigm—stay aware, adapt, change—to XP itself. Kent has revisited, cleaned-up, and refactored every bit of XP Explained and integrated many new insights. The result is XP Explained even better explained!

This is an excellent opportunity to reflect on how XP has influenced my own software development. Shortly after the first edition of XP Explained I became involved in the Eclipse project and it is now absorbing all my software energy. Eclipse isn't run under the pure XP flag. We follow agile practices; however, the XP influences are easy to spot. The most obvious one is that we have encoded several XP practices directly into our tool. Refactoring, unit testing, and immediate feedback as you code are now an integral part of our toolset. Moreover, since we are "eating our own dog food" we use these practices in our day-to-day development. Even more interesting are the XP influences one can spot in our development process. Eclipse is an open source project and one of our goals is to practice completely transparent development. The rationale is simple; if you don't know where the project is going you cannot help out or provide feedback. XP practices help us to achieve this goal.

Here is how we apply some of these practices:

- *Testing early, often and automated*—To get a green check mark for our latest builds more than 21,000 unit tests have to pass.
- *Incremental design*—We invest in the design every day, but we have the additional constraint that we need to keep our APIs stable.
- *Daily deployment*—Components deploy their code at least once per day and develop on top of the deployed code to get immediate feedback and to catch problems early.
- *Customer involvement*—We are lucky to have an active user community that isn't shy and provides us with continuous feedback. We listen and do our best to be responsive.
- *Continuous integration*—The latest code is built every night. The nightly builds provide us with insights about cross-component integration problems. Once per week we do an integration build where we ensure integrity across all components.
- *Short development cycles*—Our cycles are longer than the XP-suggested one week cycles, but the goals are the same. Each of our six week cycles ends in a milestone build which have become the heartbeat of our project. The goal of each milestone build is to show progress (which keeps us honest) and to deliver it with a high enough level of quality that our community can really use it and provide feedback (which keeps us even more honest).
- *Incremental planning*—After a release we develop an embryonic overall plan which we evolve throughout the release cycle. This plan is posted on our website early so that our user community can join the dialog. The exception is the milestones, which are fixed in the first planning iteration since they define the heartbeat of our project.

Despite the fact that we have not adopted XP in its entirety, we are getting a lot out of the above XP practices. In particular, they help us to reduce our development stress! All these practices, underpinned by a strong team committed to shipping quality software on time, are our keys to hitting the projected milestones and ship dates with precision.

Kent is continuing to challenge my views on software development. While reading the book I've discovered several practices that I will add to my try-list. I suggest you do the same and accept the XP invitation to improve the way you develop software and to create outstanding software.

Erich Gamma
September 2004

This page intentionally left blank

Foreword to the First Edition

Extreme Programming (XP) nominates coding as the key activity throughout a software project. This can't possibly work!

Time to reflect for a second about my own development work. I work in a just-in-time software culture with compressed release cycles spiced up with high technical risk. Having to make change your friend is a survival skill. Communication in and across often geographically separated teams is done with code. We read code to understand new or evolving subsystem APIs. The life cycle and behavior of complex objects is defined in test cases, again in code. Problem reports come with test cases demonstrating the problem, once more in code. Finally, we continuously improve existing code with refactoring. Obviously our development is code-centric, but we successfully deliver software in time, so this can work after all.

It would be wrong to conclude that all that is needed to deliver software is daredevil programming. Delivering software is hard, and delivering quality software in time is even harder. To make it work requires the disciplined use of additional best practices. This is where Kent starts in his thought-provoking book on XP.

Kent was among the leaders at Tektronix to recognize the potential of man in the loop pair programming in Smalltalk for complex engineering applications. Together with Ward Cunningham, he inspired much of the pattern movement that has had such an impact on my career. XP describes an approach to development that combines practices used by many successful developers that got buried under the massive literature

on software methods and process. Like patterns, XP builds on best practices such as unit testing, pair programming, and refactoring. In XP these practices are combined so that they complement and often control each other. The focus is on the interplay of the different practices, which makes this book an important contribution. There is a single goal to deliver software with the right functionality and hitting dates. While OTI's successful Just In Time Software process is not pure XP, it has many common threads.

I've enjoyed my interaction with Kent and practicing XP episodes on a little thing called JUnit. His views and approaches always challenge the way I approach software development. There is no doubt that XP challenges some traditional big M approaches; this book will let you decide whether you want to embrace XP or not.

Erich Gamma
August 1999

Preface

The goal of Extreme Programming (XP) is outstanding software development. Software can be developed at lower cost, with fewer defects, with higher productivity, and with much higher return on investment. The same teams that are struggling today can achieve these results by careful attention to and refinement of how they work, by pushing ordinary development practices to the extreme.

There are better ways and worse ways to develop software. Good teams are more alike than they are different. No matter how good or bad your team you can always improve. I intend this book as a resource for you as you try to improve.

This book is my personal take on what it is that good software development teams have in common. I've taken things I've done that have worked well and things I've seen done that worked well and distilled them to what I think is their purest, most "extreme" form. What I'm most struck with in this process is the limitations of my own imagination in this effort. Practices that seemed impossibly extreme five years ago, when the first edition of this book was published, are now common. Five years from now the practices in this book will probably seem conservative.

If I only talked about what good teams *do* I would be missing the point. There are legitimate differences between outstanding teams' actions based on the context in which they work. Looking below the surface, where their activities become ripples in the river hinting at

shapes below, there is an intellectual and intuitive substrate to software development excellence that I have also tried to distill and document.

Critics of the first edition have complained that it tries to force them to program in a certain way. Aside from the absurdity of me being able to control anyone else's behavior, I'm embarrassed to say that was my intention. Relinquishing the illusion of control of other people's behavior and acknowledging each individual's responsibility for his or her own choices, in this edition I have tried to rephrase my message in a positive, inclusive way. I present proven practices you can add to your bag of tricks.

- ✧ No matter the circumstance you can always improve.
- ✧ You can always start improving with yourself.
- ✧ You can always start improving today.

Acknowledgments

I would like to thank my most excellent group of reviewers, each of whom spent considerable time reading and commenting on the manuscript: Francesco Cirillo, Steve McConnell, Mike Cohn, David Anderson, Joshua Kerievsky, Beth Andres-Beck, and Bill Wake. The Silicon Valley Patterns Group also provided valuable feedback on drafts: Chris Lopez, John Parello, Phil Goodwin, Dave Smith, Keith Ray, Russ Rufer, Mark Taylor, Sudarsan Piduri, Tracy Bialik, Jan Chong, Rituraj Kirti, Carlos Mc Evilly, Bill Venners, Wayne Vucenic, Raj Baskaran, Tim Huske, Patrick Manion, Jeffrey Miller, and Andrew Chase. Thanks to the production staff at Pearson: Julie Nahil, Kim Arney Mulcahy, and Michelle Vincenti. Paul Petralia, my editor, saw me through difficult times with humor and understanding. He taught me lessons in the value of relationships. Erich Gamma, my pair programming partner, provided conversation and feedback. The owners and staff of Bluestone Bakery and Cafe kept the hot chocolate and broadband flowing. Joëlle Andres-Beck edited copy and collected garbage. All of my children; Lincoln, Lindsey, Forrest, and Joëlle; spent many hours at Bluestone while we edited. Gunjan Doshi provided thought-provoking questions.

Finally, I cannot possibly give sufficient thanks to my wife, developmental editor, friend, and intellectual colleague Cynthia Andres.

Chapter 1

What is XP?

Extreme Programming (XP) is about social change. It is about letting go of habits and patterns that were adaptive in the past, but now get in the way of us doing our best work. It is about giving up the defenses that protect us but interfere with our productivity. It may leave us feeling exposed.

It is about being open about what we are capable of doing and then doing it. And, allowing and expecting others to do the same. It is about getting past our adolescent surety that "I know better than everyone else and all I need is to be left alone to be the greatest." It is about finding our adult place in the larger world, finding our place in the community including the realm of business/work. It is about the process of becoming more of our best selves and in the process our best as developers. And, it is about writing great code that is really good for business.

Good relationships lead to good business. Productivity and confidence are related to our human relationships in the workplace as well as to our coding or other work activities. You need both technique and good relationships to be successful. XP addresses both.

Prepare for success. Don't protect yourself from success by holding back. Do your best and then deal with the consequences. That's extreme. You leave yourself exposed. For some people that is incredibly scary, for others it's daily life. That is why there are such polarized reactions to XP.

XP is a style of software development focusing on excellent application of programming techniques, clear communication, and teamwork which allows us to accomplish things we previously could not even imagine. XP includes:

- A philosophy of software development based on the values of communication, feedback, simplicity, courage, and respect.
- A body of practices proven useful in improving software development. The practices complement each other, amplifying their effects. They are chosen as expressions of the values.
- A set of complementary principles, intellectual techniques for translating the values into practice, useful when there isn't a practice handy for your particular problem.
- A community that shares these values and many of the same practices.

XP is a path of improvement to excellence for people coming together to develop software. It is distinguished from other methodologies by:

- Its short development cycles, resulting in early, concrete, and continuing feedback.
- 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, customers, and testers 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 actively engaged individuals with ordinary talent.
- Its reliance on practices that work with both the short-term instincts of the team members and the long-term interests of the project.

The first edition of *Extreme Programming Explained: Embrace Change* had a definition of XP with the advantage of clarity: "XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements." While this statement was true about the origin and intent of XP, it doesn't tell the whole story. In the five years since the publication of the first edition teams have pushed XP much further than the original definition. XP can be described this way:

- XP is lightweight. In XP you only do what you need to do to create value for the customer. You can't carry a lot of baggage and move fast. However, there is no freeze-dried software process. The body of technical knowledge necessary to be an outstanding team is large and growing.
- XP is a methodology based on addressing constraints in software development. It does not address project portfolio management, financial justification of projects, operations, marketing, or sales. XP has implications in all of these areas, but does not address these practices directly. Methodology is often interpreted to mean "a set of rules to follow that guarantee success." Methodologies don't work like programs. People aren't computers. Every team does XP differently with varying degrees of success.
- XP can work with teams of any size. Five years ago, I did not want to claim too much. Others have since put XP to use in a wide range of projects and have had success with both large and small projects and teams. The values and principles behind XP are applicable at any scale. The practices need to be augmented and altered when many people are involved.
- XP adapts to vague or rapidly changing requirements. XP is still good for this situation, which is fortunate because requirements need to change to adapt to rapid shifts in the modern business world. However, teams have also successfully used XP where requirements don't seem volatile, like porting projects.

XP is my attempt to reconcile humanity and productivity in my own practice of software development and to share that reconciliation. I had begun to notice that the more humanely I treated myself and others,

the more productive we all became. The key to success lies not in self-mortification but in acceptance that we are people in a person-to-person business.

Technique also matters. We are technical people in a technical field. There are better ways and worse ways of working. The pursuit of excellence in technique is critical in a social style of development. Technique supports trust relationships. If you can accurately estimate your work, deliver quality the first time, and create rapid feedback loops; then you can be a trustworthy partner. XP demands that participants learn a high level of technique in service of the team's goals.

XP means giving up old habits of working for new ways tailored to today's reality. The habits, attitudes, and values of our early years worked then; but may not be our best choices in the current world of team software development. Good, safe social interaction is as necessary to successful XP development as good technical skills.

One example is the concept that vulnerability is safety. The old habit of holding something back in order to be safe doesn't really work. Holding back that last 20% of effort doesn't protect me. When my project fails, the fact that I didn't give my all doesn't actually make me feel better. It doesn't protect me from a sense of failure that I couldn't make the project work. If I do my very best writing a program and people don't like it, I can still feel justly good about myself. This attitude allows me to feel safe no matter the circumstance. If how I feel is based on an accurate read on whether I did my best, I can feel good about myself by doing my best.

XP teams play full out to win and accept responsibility for the consequences. When self-worth is not tied to the project, we are free to do our best work in any circumstance. In XP you don't prepare for failure. Keeping a little distance in relationships, holding back effort either through underwork or overwork, putting off feedback for another round of responsibility diffusion: none of these behaviors have a place on an XP team.

You may have enough time, money, or skills on your team or you may not; but it is always best to act as if there is going to be enough. This "mentality of sufficiency" is movingly documented by anthropologist Colin Turnbull in *The Mountain People* and *The Forest People*. He contrasts two societies: a resource-starved tribe of lying, cheating backstabbers and a resource-rich, cooperative, loving tribe. I often ask developers

in a dilemma, "How would you do it if you had enough time?" You can do your best work even when there are constraints. Fussing about the constraints distracts you from your goals. Your clear self does the best work no matter what the constraints are.

If you have six weeks to get a project done, the only thing you control is your own behavior. Will you get six weeks' worth of work done or less? You can't control others' expectations. You can tell them what you know about the project so their expectations have a chance of matching reality. My terror of deadlines vanished when I learned this lesson. It's not my job to "manage" someone else's expectations. It's their job to manage their own expectations. It's my job to do my best and to communicate clearly.

XP is a software development discipline that addresses risk at all levels of the development process. XP is also productive, produces high-quality software, and is a lot of fun to execute. How does XP address the risks in the development process?

- Schedule slips—XP calls for short release cycles, a few months at most, so the scope of any slip is limited. Within a release, XP uses one-week iterations of customer-requested features to create fine-grained feedback about progress. Within an iteration, XP plans with short tasks, so the team can solve problems during the cycle. Finally, XP calls for implementing the highest priority features first, so any features that slip past the release will be of lower value.
- Project canceled—XP asks the business-oriented part of the team to choose the smallest release that makes the most business sense, so there is less to go wrong before deploying and the value of the software is greatest.
- System goes sour—XP creates and maintains a comprehensive suite of automated tests, which are run and rerun after every change (many times a day) to ensure a quality baseline. XP always keeps the system in deployable condition. Problems are not allowed to accumulate.
- Defect rate—XP tests from the perspective of both programmers writing tests function-by-function and customers writing tests program-feature-by-program-feature.
- Business misunderstood—XP calls for business-oriented people to be first-class members of the team. The specification of the project

is continuously refined during development, so learning by the customer and the team can be reflected in the software.

- Business changes—XP shortens the release cycle, so there is less change during the development of a single release. During a release, the customer is welcome to substitute new functionality for functionality not yet completed. The team doesn't even notice if it is working on newly discovered functionality or features defined years ago.
- False feature rich—XP insists that only the highest priority tasks are addressed.
- Staff turnover—XP asks programmers to accept responsibility for estimating and completing their own work, gives them feedback about the actual time taken so their estimates can improve, and respects those estimates. The rules for who can make and change estimates are clear. Thus, there is less chance for a programmer to get frustrated by being asked to do the obviously impossible. XP also encourages human contact among the team, reducing the loneliness that is often at the heart of job dissatisfaction. Finally, XP incorporates an explicit model of staff turnover. New team members are encouraged to gradually accept more and more responsibility, and are assisted along the way by each other and by existing programmers.

XP assumes that you see yourself as part of a team, ideally one with clear goals and a plan of execution. XP assumes that you want to work together. XP assumes that change can be made inexpensive using this method. XP assumes that you want to grow, to improve your skills, and to improve your relationships. XP assumes you are willing to make changes to meet those goals.

Now I'm ready to answer the question posed by this chapter: what is XP?

- XP is giving up old, ineffective technical and social habits in favor of new ones that work.
- XP is fully appreciating yourself for total effort today.
- XP is striving to do better tomorrow.

- ✧ XP is evaluating yourself by your contribution to the team's shared goals.
- ✧ XP is asking to get some of your human needs met through software development.

The rest of this book explores what to do to effect these changes and speculates about why they work, personally and economically. The book is divided into two sections. The first is practical, describing a way of doing and thinking about software development that both assumes and satisfies human needs, including the need for relationships. The second section covers the philosophical and historical roots of XP and places XP in today's context.

There are as many ways of reading this book and applying XP as there are of getting into a cool pool on a hot day: one toe at a time, walking steadily down the steps, the cannonball, the racing dive. They all meet the goal of getting into the water. Your choice may be based on style, speed, efficiency, or fear. Only you can decide which is right for you. I hope that in reading and applying this book you will come to a deeper understanding of why you are involved in software development and how you can find satisfaction from this work.

This page intentionally left blank

Index

D