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

but I haven't tried it. I like to program with someone new every couple of hours, switching at natural breaks in development.

Pairing and Personal Space

An issue that has come up and requires comment is the close contact in pair programming. Different individuals and cultures are comfortable with different amounts of body space. Pairing with an Italian who communicates best when very close is completely different than pairing with a Dane who likes a few feet of personal space. If you aren't aware of the difference it can be acutely uncomfortable. Personal space must be respected for both parties to work well.

Personal hygiene and health are important issues when pairing. Cover your mouth when you cough. Don't come to work when you are sick. Avoid strong colognes that might affect your partner.

Working effectively together feels good. It may be a new experience in the workplace for some. When programmers aren't emotionally mature enough to separate approval from arousal, working with a person of the opposite gender can bring up sexual feelings that are not in the best interest of the team. If these feelings arise when pairing, stop pairing with the person until you have taken responsibility for and dealt with your feelings. Even if the feelings are mutual, acting on them will hurt the team. If you want to have an intimate relationship, one of you should leave the team so you can build a personal relationship in a personal setting without confusing the team's communication with a sexual subtext. Ideally, emotions at work will be about work.

It is important to respect individual differences when pairing. In Figure 6 the man has moved closer to the woman than is comfortable for her. Neither is making his or her best technical decisions at this point, although they may be completely unaware of the source of their discomfort.

If you are uncomfortable pairing with someone on the team, talk about it with someone safe; a respected team member, a manager, or someone in human resources. If you aren't comfortable, the team isn't doing as well as it could. And chances are others are uncomfortable too.

FIGURE 6. Personal space and pairing

Stories

Plan using units of customer-visible functionality. "Handle five times the traffic with the same response time." "Provide a two-click way for users to dial frequently used numbers." As soon as a story is written, try to estimate the development effort necessary to implement it.

Software development has been steered wrong by the word "requirement", defined in the dictionary as "something mandatory or obligatory." The word carries a connotation of absolutism and permanence, inhibitors to embracing change. And the word "requirement" is just plain wrong. Out of one thousand pages of "requirements", if you deploy a system with the right 20% or 10% or even 5%, you will likely realize all of the business benefit envisioned for the whole system. So what were the other 80%? Not "requirements"; they weren't really mandatory or obligatory.

Early estimation is a key difference between stories and other requirements practices. Estimation gives the business and technical per-

spectives a chance to interact, which creates value early, when an idea has the most potential. When the team knows the cost of features it can split, combine, or extend scope based on what it knows about the features' value.

Give stories short names in addition to a short prose or graphical description. Write the stories on index cards and put them on a frequently-passed wall. Figure 7 is a sample card of a story I wish my scanner program implemented. Every attempt I've seen to computerize stories has failed to provide a fraction of the value of having real cards on a real wall. If you need to report progress to other parts of the organization in a familiar format, translate the cards into that format periodically.

One feature of XP-style planning is that stories are estimated very early in their life. This gets everyone thinking about how to get the greatest return from the smallest investment. If someone asks me whether I want the Ferrari or the minivan, I choose the Ferrari. It will inevitably be more fun. However, as soon as someone says, "Do you want the Ferrari for $150,000 or the minivan for $25,000?" I can begin to make an informed decision. Adding new constraints like "I need to haul five children" or "It has to go 150 miles per hour" clear the picture further. There are cases where either decision makes sense. You can't make a good decision based on image alone. To choose a car wisely you need to know your constraints, both cost and intended use. All other things being equal, appeal comes into play.

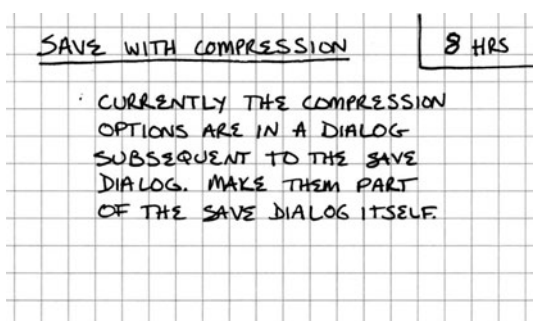

FIGURE 7. Sample story card

Weekly Cycle

Plan work a week at a time. Have a meeting at the beginning of every week. During this meeting:

1. Review progress to date, including how actual progress for the previous week matched expected progress.
2. Have the customers pick a week's worth of stories to implement this week.
3. Break the stories into tasks. Team members sign up for tasks and estimate them.

Start the week by writing automated tests that will run when the stories are completed. Then spend the rest of the week completing the stories and getting the tests to pass. A team proud of its work will fully implement the stories, not just do enough work to get the tests to pass. The goal is to have deployable software at the end of the week which everyone can celebrate as progress.

The week is a widely shared time scale. The nice thing about a week as opposed to two or three (as I recommended in the first edition) is that everyone is focused on Friday. The team's job—programmers, testers, and customers together—is to write the tests and then get them to run in five days. If you get to Wednesday and it is clear that all the tests won't be running, that the stories won't be completed and ready to deploy, you still have time to choose the most valuable stories and complete them.

Some people like to start their week on a Tuesday or Wednesday. I was surprised when I first saw it, but it's common enough to mention. As one manager told me, "Monday's are unpleasant and planning is unpleasant, so why put them together?" I don't think planning should be unpleasant; but in the meantime, shifting the start of the cycle makes some sense as long as it doesn't put pressure on people to work over the weekend. Working weekends is not sustainable; if the real problem is that the estimates are overly optimistic, then work on improving your estimates.

Planning is a form of necessary waste. It doesn't create much value all by itself. Work on gradually reducing the percentage of time you spend planning. Some teams start with a whole day of planning for a

week, but gradually refine their planning skills until they spend an hour planning for the week.

I like to break stories into tasks that individuals take responsibility for and estimate. I think ownership of tasks goes a long way towards satisfying the human need for ownership. I've seen other styles work well, though. You can write small stories that eliminate the need for separate tasks. The cost of this approach is more work for the customer. You can also eliminate sign-up by keeping a stack of tasks. When a programmer is ready to start a new task, he takes one from the top of the stack. This eliminates the opportunity for him to choose a task he particulary cares about or is especially good at, but ensures that each programmer gets a variety of tasks. (Pairing gives programmers a chance to use specialist skills, whoever holds the task.)

The weekly heartbeat also gives you a convenient, frequent, and predictable platform for team and individual experiments. "OK, for the next week we're going to switch pair partners every hour on the hour," or "I'll juggle for five minutes every morning before I start programming."

Quarterly Cycle

Plan work a quarter at a time. Once a quarter reflect on the team, the project, its progress, and its alignment with larger goals.

During quarterly planning:

- ✧ Identify bottlenecks, especially those controlled outside the team.
- ✧ Initiate repairs.
- ✧ Plan the theme or themes for the quarter.
- ✧ Pick a quarter's worth of stories to address those themes.
- ✧ Focus on the big picture, where the project fits within the organization.

A season is another natural, widely shared timescale to use in organizing time for a project. Using a quarter as a planning horizon synchronizes nicely with other business activities that occur quarterly. Quarters are also a comfortable interval for interaction with external suppliers and customers.

The separation of "themes" from "stories" is intended to address the tendency of the team to get focused and excited about the details of

what they are doing without reflecting on how this week's stories fit into the bigger picture. Themes also fit well into larger-scale planning such as drawing marketing roadmaps.

Quarters are also a good interval for team reflection, finding gnawing-but-unconscious bottlenecks. You can also propose and evaluate long-running experiments quarterly.

Slack

In any plan, include some minor tasks that can be dropped if you get behind. You can always add more stories later and deliver more than you promised. It is important in an atmosphere of distrust and broken promises to meet your commitments. A few met commitments go a long way toward rebuilding relationships.

In Iceland, one of the winter sports is taking monstrous trucks bouncing around the backcountry. These trucks all have four-wheel-drive; but when they are out crashing around, they only use two-wheel-drive. If they get stuck in two-wheel-drive they have four-wheel-drive to get them out. If they get stuck in four-wheel-drive they're just stuck.

I remember two conversations: one with a middle manager who had one hundred people reporting to him and another with his executive manager who had three hundred people in his organization. I suggested to the middle manager that he encourage his teams to only sign up for what they were confident they could actually do. They had a long history of overcommitting and underdelivering. "Oh, I couldn't do that. If I don't agree to aggressive [i.e. unrealistic] schedules, I'll be fired." The next day, I talked to the executive. "Oh, they never come in on time. It's okay. They still deliver enough of what we need."

I had been watching first-hand the incredible waste generated by their habitual overcommitment: unmanageable defect loads, dismal morale, and antagonistic relationships. Meeting commitments, even modest ones, eliminates waste. Clear, honest communication relieves tension and improves credibility.

You can structure slack in many ways. One week in eight could be "Geek Week". Twenty percent of the weekly budget could go to programmer-chosen tasks. You may have to begin slack with yourself, telling yourself how long you actually think a task will take and giving yourself time to do it, even if the rest of the organization is not ready for honest and clear communication.