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.