Getting Started with XP: Toe Dipping, Racing Dives, and Cannonballs

Kent Beck, Three Rivers Institute
Cynthia Andres, Three Rivers Institute

> I've read about Extreme Programming and it sounds interesting. It seems so far from where we are today, though. Where do I start? How do I start?

The recently published second edition of *Extreme Programming Explained: Embrace Change* explains how and why to use XP. But where to start? *XP Explained* uses the analogy of entering a swimming pool to describe how organizations get started with XP. There are toe dippers, racing divers, cannonballers, and all manner of variations in between. In this paper I'll characterize these styles. I hope you will find a style of applying XP that works for you and your organization.

Here are some of the promised benefits of XP: fewer defects, more predictability, greater flexibility, closer conformance between delivered features and actual needs, and shorter lead time for new features. A typical XP project has weekly milestone releases, each of which is technically ready to deploy. In order to be constantly ready to deploy, the system is being continuously exercised by automated tests, some written by programmers to avoid debugging time and some written by testers and business experts on the team. The whole team; programmers, testers, project and product managers, real customers, and analysts; sits together in an open workspace festooned with up-to-date project information (my favorite being a lava lamp that glows green if the most recent build passed all its tests). Projects requiring more people than fit comfortably in one room are broken into sub-projects which are coordinated conventionally or using a process like Scrum that calls for monthly milestones. This style is similar to how some existing teams work, but it is very different from how others work.

A problem that people have had since the articulation of XP is how to how to start moving from how they work today toward an XP style of development. You want to start in the right place, convince the right people, and "do it right" so you can reap all the benefits. There is no "right". There is just the doing or the freezing in fear. XP is about courage, about embracing the experience of change. It starts when you start and where you start. Eight years of observing and coaching the application of XP has convinced me that there are as many ways of starting and sustaining change as there are ways to get into a swimming pool.

In spite of diversity in applying XP, there are some common factors to successfully instigating change.

1. Read XP Explained, 2nd edition. This will give you a shared vocabulary for the techniques you are about to try.
2. Share what you have learned with others. Change happens best with the support of a like-minded community.
3. Make a public commitment to change. Calling your shot in public is a great motivator to stick with change when it gets hard.

4. Make a plan for your changes. XP-style planning is a good way to prioritize when you have too many changes to make all at once. Start with the area you can best leverage.

The XP principles that guide our choice of practices for software development come into play when starting with XP. In addition, there are some other principles to consider regarding how we work and institute change in the workplace:

- Human speed—people can only change so fast. When that speed is exceeded, they revert.
- Self-interest—people need to see why the changes are in their best interest.
- Grow or die—change spreads through an organization. If not, those who have changed will be forced to recant or leave.
- Safety—people need to feel safe to change. The organization needs to support change, through both the inevitable hiccups and the successes. Safety is relative, though, so the risks of the change process may seem safer than a disaster they know is coming.

Applying these principles in different organizations and in different circumstances will result in very different styles of change. Here are three stereotypical ways teams and organizations get started with XP, though the "right" way to start XP for your company is the one you think will work for you.

Toe Dipping

Some people and teams value continuity. They don't want to let go with one hand before they have a firm hold with the other. When they get started with XP they start one practice at a time, firmly instilling one before adding the next, and leaving the rest of their development process intact in the meantime. A few years ago I thought that teams would start with either testing or planning. Experience has shown that the gradual path into XP has many entrances. That's why the second edition of XP Explained describes 13 "primary" practices (listed in the appendix), all of which are safe places to start and any of which will provide some immediate improvement.

Examples of toe dipping include making a point of programming together in a conference room several hours a day, having developers write some automated tests as they code, or even just dividing a big risky release into two smaller releases. Some toe dippers work on XP on their own. Individuals can start improving their own process even if the whole team is not yet ready. Others hire coaches to help facilitate the process.

If you are a toe dipper, think about the area you would most like to improve, find the practice that addresses that issue, and implement it on a trial basis. After a month or two, evaluate the effects of this change, barriers you met, successes you had, and share your experiences with your support community. Then refine or repeat the process to add the next most valuable practices.

Cannonball

Some teams want to feel in control of the changes to their software process. They want quick results and they are willing to deal with the chaos of radical change as long as they are in control. These are the teams that start doing every practice they can at full throttle.

The result is, predictably for the short term, chaos; but it can be constructive chaos. Everyone is learning new techniques every day and those techniques interact in unpredictable ways. Some days work smoothly. Others days are like driving in bumper-to-bumper traffic. After a sharp learning curve, things settle out into a much more highly interactive and refined team able to leap small buildings in a single bound. They produce more, make fewer mistakes, and have confidence in themselves and their ability to handle challenges.

The cannonball is attractive when you want a fresh start and there aren't any catastrophic effects from the ensuing chaos. If you are beginning a release planned to take nine months, for instance, a cannonball might be a good choice. It is not a good choice if you have two weeks left before deployment. Amplified positive interactions between practices, quick turnaround, and the confidence boost the team gets from gaining control on their own are all good reasons to try a cannonball.

Some of chaos of the cannonball is mitigated by the synergies between the practices. If you are going from an eight-week testing phase to a one-week testing phase, you would be sunk without the tests written by the programmers.

One of the challenges of managing the cannonball is that groups outside the team are quickly affected by the team's changes. The team will ask for communication sooner and more directly than they used to. They will likely break existing power chains, skipping across the organizational chart to find the information they need to succeed. Whether a cannonball results in lasting improvement depends not only on how the team does its work but also on how the rest of the organization responds to their change. Outside support can encourage a team to stick with their changes long enough to see improvement and form new habits. Executive support for the change is essential for breaking organizational log jams. Otherwise you'll have a frustrated team unable to grow because they can't get the help they need.

Teams that cannonball successfully have a sense of pride and confidence in their accomplishments and flexibility. They know they can adapt to whatever circumstances they encounter.

Racing Dive

Teams that want quick results and are willing to trust outsiders often turn to XP coaches for an "assisted cannonball". A good coach can smooth out the rough edges of your entry and save you some of the pain of learning. The name of the services provided by some coaches, such as "immersion", suggest the experience—dive in and do it. These teams make a lot of changes quickly but they have the support of someone who has been there before and has the experience to eliminate some of the pain and accelerate the learning.

The racing dive is a good choice for a team that wants quick results but can't afford as much floundering and chaos as the cannonballers. It is also a good choice for teams that want the effects of XP but don't have the courage or persistence to make and sustain change on their own.

One of the limitations of learning XP on your own is that teams sometimes can't even imagine working in the XP style. I have pair programmed with self-taught XPers for whom test-first programming and refactoring were a revelation. They didn't realize just how tiny the steps could be, how many tests could profitably be written, and how often they could be run.

Finding Your Style

Each of these strategies has its place. Since people have very different appetites for change, a team may not even agree on which strategy there are using. One man's toe dipping is another man's cannonball. Here are some questions to help you find a style and pace of change that will meet your needs. Taking the time to think about and discuss these issues will smooth your transition.

- How fast does the team need to see results?
- How dramatic do those results need to be?
- How much does the organization have to spend on outside help?
- How strong are the relationships within the team and between the team and the rest of the organization?

Regardless of the style you choose, beware of the changing/changing back phenomenon. Any time you make a change, it puts a strain on surrounding people, resources, and organizations related to that change. It takes awareness and persistence to stick with a change and follow through on its consequences until the new behavior becomes habit in spite of pressure from those around you to make their lives "easier" by changing back.

An antidote to changing back is being accountable to a supportive community. If the whole team decides that they are going to integrate their changes every hour and how often people integrate is public knowledge, it's much easier to keep integrating often even if it feels uncomfortable or seems inconvenient. Even if you are the only one making a change you will find it easier if you participate in an online community or attend your local area user's group meeting. In time, rhythm replaces strength.

Where you need to go to get support will be very different depending on your position in the organization and where the impetus for change began. In the early days of XP, almost all those wanting to apply it were programmers out to improve their own work. Project managers and customers had little motivation to change their style of work until the programmers had demonstrated improvement. In this case, programmers had to get started with the parts of XP that were purely technical: test-first programming, pair programming, continuous integration, incremental design.

If you are a programmer wanting support for changing your style of work, show why the change is good both for you and for those from whom you want support. Share the benefits you know about and what you have learned from others. Get a commitment for a trial. When you have tried the change, report back to your supporters about how it went and what you would like to do next.

Increasingly, it is the business sponsors who are asking for XP because they want to be able to see clearly into their projects, to detect problems early, and to be able to manage the scope of the system as it evolves. The first practices to begin with for customers are those related to planning: weekly and quarterly planning, stories, and slack. To improve on-time deliveries, the technical people on the team will need to begin applying the technical practices, but establishing a shared understanding of the evolving scope of the system can be a big step towards working together.

Some people are in the position of being forced to apply XP. If this describes you, you'll need to decide how you are going to treat XP. You can resist or you can make XP your own. Here is the case for trying XP in spite of external pressure to do so. One of the

principles in XP is mutual benefit. As Gandhi said, “Only that solution is just that is in the best interest of all parties.” Changing to XP will likely be uncomfortable at times, but it should also serve your interests as well as those of the team and the whole organization. Take advantage of the opportunity to use XP to improve your own work and practice. Take a look at the practices in the appendix. If one of them reminds you of times you’ve been especially effective at software development, embrace it. Try it out yourself. Find a buddy to try it with. Convince the team to try it early. The result will be a change process that intentionally meets your needs as well, instead of one driven solely by other people’s agendas.

Whatever your circumstances; whether you go in fast or slow, whether you make a big splash or just a few ripples, whether you have help or not; XP has something to offer you. Now is the time to get started. Where is your place to start? Find a style of improvement that suits you and start the process today. Software development has just begun to create value in business. These improvements are available to you as soon as you begin applying XP.

Appendix: Primary Practices

The following is a short summary of the primary practices in XP. Practices are not rules to be slavishly followed. They are ideas for improvement. You and your team may revisit a practice many times as you learn. For example, the first time you look at Pair Programming you may decide to just try weekly code reviews. After a while you may try real pairing, but only for a limited time. With more experience you may try remote pairing.

I recommend going through this list and drawing a picture of what each practice brings to mind. If one practice strikes you, try it first. Try it for a few weeks and then evaluate it. Is it something you want to continue? Modify? Drop?

- Whole Team—the team includes people with all the skills and connections it needs to succeed.
- Sit Together—the team sits within eye contact of each other.
- Pair Programming—sit two to a machine so programming becomes a conversation.
- Informative Workspace—plaster the walls with up-to-date information about the project.
- Weekly Planning—plan for visible, valuable progress each week.
- Quarterly Planning—set quarterly themes to be addressed by the weekly iterations.
- Slack—include some optional items in any schedule.
- Test-first Programming—code by writing a failing test, then making the system satisfy the test.
- Incremental Design—invest in the design only what is needed to comfortably support today's stories.
- Stories—plan and track in increments of business functionality.
- Ten-minute Build—automatically build and test as much of the system as you can in ten-minutes.
- Continuous Integration—integrate your changes with the shared code every couple of hours at most.
- Energized Work—work and live so you can bring energy to your work.