

XP—An Overview

Abstract

Extreme Programming (XP) and the other “Agile Methods” have recently burst onto the scene, and many people are talking and asking lots of questions. This article is designed for the uninitiated, to provide a minimal understanding of XP. It provides the basic understanding of XP you may need to understand the other articles on this web site that discuss XP.

Uses of XP

The author of XP, Kent Beck is explicit that it is designed for use in exploratory projects. That is, it is best suited for projects where the end results are only vaguely known at the start of the project, or there are technological or user-interface questions that the project is supposed to answer. It is also aimed at projects that are characterized by a small co-located development team of up to 10 people.

XP Values

In order to understand the XP practices, we must first look at the Values on which they are based. Everything in the practices must be interpreted in the light of these four Values.

Communication. Face-to-face, person-to-person communication is critical to XP, and allows documentation to be kept to a minimum. In the volatile environments for which XP was designed, this is important for two reasons.

First, documentation is *not* equivalent to communication. A document contains words and pictures that must be interpreted, and different people’s interpretations of the same words and pictures are guaranteed to differ. Face-to-face communication is much richer than the written word, because it includes intonation, gestures, body language and a host of other information. XP does not dispense with documentation, rather it relegates it to a place that is secondary to inter-personal communication. The purpose of documentation is to stimulate memory, not to be the primary mode of communication.

Second, in an environment of uncertainty and change, all written documents will become obsolete in short order. The less documentation there is, the less onerous is the task of keeping it up-to-date and relevant.

Simplicity. XP asks, “*What is the simplest thing that could possibly work?*” This question applies equally to software design, process definition, status tracking and anything else in which the project must engage. The main idea is that, in an environment that is characterized by the expectation of change, anything you do may have to be reworked tomorrow. Trying to anticipate the unknowable is wasted effort, so you should spend only the effort that is required to meet the current need. This may not be optimal, but in a change-filled project, *nothing* is optimal.

Feedback. Kent Beck (the author of XP) tells us, “*Optimism is an occupational hazard of programming. Feedback is the treatment.*” Many of XP’s practices embody this principle of quick and continuous feedback. This includes feedback from:

- The customer – on the utility of what is being built,
- Your peers – on the quality of your work, and
- The computer – on how well everything is integrating.

XP—An Overview

Courage. This refers to the courage to do the “right thing” for the current project, even when it contradicts conventional wisdom, or is different from what we’ve done before. This would include not only the courage to use XP for the first time, but also the courage to question XP’s practices when they don’t seem to work on your project.

These four values of Communication, Simplicity, Feedback and Courage are the basis upon which the practices of XP are built, and the guide by which they are interpreted.

XP Practices

XP is defined by its 12 practices. They are:

The Planning Game. Jim Highsmith defines the planning game as, “... the point of interaction between customers and developers, specifying the responsibilities of both parties.” This “game” is played at the beginning of each iteration (see “Small Releases”, below for more information on these iterations). This game consists of a variety of activities that are all necessary to assure the project has a good footing on which to proceed with the next iteration. Based on all that has been learned by both the developers and the customer in the previous iterations, these things are done:

- Update the “Metaphor” (see “Metaphor”, below).
- Update, add and delete “Stories”. The requirements for each feature of the system are defined in a “Story”. Each Story describes its feature from the customer’s perspective in short concise prose (fits on a 3x5 card). (Keeping in mind the “Communication” value describe above, the Story is worked out face-to-face between the customer and the developers. What is written on the 3x5 card serves only as a memory of the story.)
- Revise the estimates of the project’s scope, cost and schedule.
- Revise the Release Plan (which features will be produced in each iteration). This includes agreeing on exactly what will be included in this next iteration.

Small Releases. XP projects are defined as a series of short iterations (of about 3 weeks), each of which results in a software “release” that has some utility or value from the customer’s perspective. From a practical perspective, many of the releases would never be installed in a customer’s operation. But *every* release must contain one or more new or updated features that the customer can test, learn from, and give feedback to the developers on.

Metaphor. The Metaphor of the system is a broad conceptual idea of the general results that the project is to produce. Because of the experimental nature of projects for which XP was designed, a concise and explicit specification of the expected system is not useful, and often not possible. So, rather than waste effort on such a specification, an XP project defines and maintains its Metaphor. The Metaphor grows and matures along with the product, and its entire shape and details are not expected to be known until the project is complete. But even in its incomplete state, it serves as an important touchstone for guiding the project activities and understanding the Story for each feature.

Simple Design. Kent Beck (the author of XP) wrote, “*If you believe that the future is uncertain, and you believe that you can cheaply change your mind, then putting in functionality on speculation is crazy.*” By its very nature, an XP project is uncertain. We start with a vague idea of what is to be accomplished, and we learn about it as we take each small step. (The idea of cheaply changing your mind is covered by the next practice, “Refactoring”.)

XP—An Overview

XP's "Simple Design" practice is simple. Design only what you need for the current iteration, and do it in the simplest way you can. Do not try to anticipate what you will need for future iterations, because the feature "Stories", the product "Metaphor", and even the entire architecture of the system could change before you get there. Anticipatory design is wasteful in an uncertain environment – don't do it!

Refactoring. Refactoring is essentially continuous incremental re-design of the system. In an XP project, we expect to be reworking previously complete code in each iteration. As any maintenance programmer knows, continual code changes without refactoring results in degraded and brittle programs that become less reliable and more difficult to change over time.

By focusing on Refactoring, XP assures the quality and reliability of the programs. It assures that as each change is made, the prior design decisions and assumptions are examined, and when appropriate, changed. Refactoring takes more time and effort than just slamming a change into a program, but it saves time in the long run by maintaining the pliability of the code.

Testing. XP's perspective on testing is different from (and much healthier than) any other I have seen. Not only is testing defined as a critical part of the developer's job, it is his or her *first* job. That is, before a developer writes a single line of code, he or she writes the unit test plan!

This concept of thinking about testing first has a tremendous beneficial impact, because before the developer begins to wrestle with making the program work, he or she takes the time to think about how it could *break!* This minor change in the thought-process results in much higher quality code, because it is written with failure in mind.

Pair Programming. XP is probably best known for this unique practice. "Pair Programming" refers to two people always working together in any software development activity (other than the Planning Game). Whether they are designing a module, writing a Unit Test Plan, coding or testing, one developer is working the keyboard, and the other is watching over the first's shoulder.

Pair Programming is characterized as the ultimate in Peer Reviews or Inspections, and it takes to the extreme the foundational principle of removing defects as early as possible. Indeed, as soon as a line is typed, the other person is evaluating it. This practice is supposed to embody the best results of all of the peer review methods—defect removal, knowledge sharing, and learning from each other. On top of that, it will result in better design and coding decisions as the pair stops and discusses options on an ongoing basis.

Collective Ownership. In most shops, each code module is "owned" by a specific programmer, and all changes are done by that one person (or at least approved by him or her). XP demands the opposite; that no one *owns* any program. Rather, anyone on the team may check out and change any program in the system as needed to complete his or her work.

XP proponents claim that this practice does not result in chaos. Rather, it yields a greater level of collaboration than is normally seen on software teams. Everyone feels responsible for the correct functioning of every module in the system, and they tend to work together to achieve that end.

Continuous Integration. Daily builds are not good enough for XP practitioners. Instead they perform many builds each day. And normally, an automated test script is run after each build to assure that some basic level of functionality continues to work. The whole point behind

XP—An Overview

this practice is to shorten the Feedback loop, so that if any change does not integrate with the rest of the system, the problem is highlighted and fixed as soon as possible.

40-Hour Week. XP is designed to support a “sustainable” development environment. That is, one that can continue working effectively on a long-term basis. Many shops have regular “crunch” times, when everyone is required to work overtime in order to achieve the project goals. This practice specifies that such crunches should be rare if they occur at all.

However, it does *not* mean that no one is allowed to work more than 40 hours per week. In fact, it is pointed out that in a sustainable environment, developers are often self-motivated to work overtime. But because this overtime is self-motivated, it does not result in burn-out, and so, it is sustainable.

On-site Customer. XP requires ongoing and regular face-to-face interaction between the development team and the customer. The level of interaction that is called for almost *demands* that the customer be co-located with the development team. An on-site customer not only makes the Planning Game and Testing much easier, it also allows the developers to have their questions answered as they go about their work, resulting in software that is more likely to satisfy the customer’s needs.

Coding Standards. This last practice is required to support several of the others (like Pair Programming and Collective Ownership). The team must all agree on the standards that will be used in the code so that any member of the team can read any modify any other team member’s work at any time.

Summary

Extreme Programming (XP) is the best known of the “Agile Methods”. It is based on these four Values:

- Communication
- Simplicity
- Feedback
- Courage

And it embodies these 12 Practices:

- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Refactoring
- Testing
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-Hour Week
- On-site Customer
- Coding Standards