agilealliance.org

What is Extreme Programming (XP)?

14. Jun 2017 2.820 words

Definition

Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

When Applicable

The general characteristics where XP is appropriate were described by Don Wells on www.extremeprogramming.org:

- Dynamically changing software requirements
- Risks caused by fixed time projects using new technology
- Small, co-located extended development team
- The technology you are using allows for automated unit and functional tests

Due to XP's specificity when it comes to it's full set of software engineering practices, there are several situations where you may not want to fully practice XP. The post When is XP Not Appropriate on the C2 Wiki is probably a good place to start to find examples where you may not want to use XP.

While you can't use the entire XP framework in many situations, that shouldn't stop you from using as many of the practices as possible given your context.

Values

The five values of XP are communication, simplicity, feedback, courage, and respect

and are described in more detail below.

Communication

Software development is inherently a team sport that relies on communication to transfer knowledge from one team member to everyone else on the team. XP stresses the importance of the appropriate kind of communication – face to face discussion with the aid of a white board or other drawing mechanism.

Simplicity

Simplicity means "what is the simplest thing that will work?" The purpose of this is to avoid waste and do only absolutely necessary things such as keep the design of the system as simple as possible so that it is easier to maintain, support, and revise. Simplicity also means address only the requirements that you know about; don't try to predict the future.

Feedback

Through constant feedback about their previous efforts, teams can identify areas for improvement and revise their practices. Feedback also supports simple design. Your team builds something, gathers feedback on your design and implementation, and then adjust your product going forward.

Courage

Kent Beck defined courage as "effective action in the face of fear" (Extreme Programming Explained P. 20). This definition shows a preference for action based on other principles so that the results aren't harmful to the team. You need courage to raise organizational issues that reduce your team's effectiveness. You need courage to stop doing something that doesn't work and try something else. You need courage to accept and act on feedback, even when it's difficult to accept.

Respect

The members of your team need to respect each other in order to communicate with each other, provide and accept feedback that honors your relationship, and to work together to identify simple designs and solutions.

Practices

The core of XP is the interconnected set of software development practices listed below. While it is possible to do these practices in isolation, many teams have found some practices reinforce the others and should be done in conjunction to fully eliminate the risks you often face in software development.

The XP Practices have changed a bit since they were initially introduced.The original twelve practices are listed below. If you would like more information about how these practices were originally described, you can visit http://ronjeffries.com/xprog/what-is-extreme-programming/.

- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-hour week
- On-site Customer
- Coding Standard

Below are the descriptions of the practices as described in the second edition of Extreme Programming Explained Embrace Change. These descriptions include refinements based on experiences of many who practice extreme programming and reflect a more practical set of practices.

Sit Together

Since communication is one of the five values of XP, and most people agree that face to face conversation is the best form of communication, have your team sit together in the same space without barriers to communication, such as cubicle walls.

Whole Team

A cross functional group of people with the necessary roles for a product form a single team. This means people with a need as well as all the people who play some part in satisfying that need all work together on a daily basis to accomplish a specific outcome.

Informative Workspace

Set up your team space to facilitate face to face communication, allow people to have some privacy when they need it, and make the work of the team transparent to each other and to interested parties outside the team. Utilize Information Radiators to actively communicate up-to-date information.

Energized Work

You are most effective at software development and all knowledge work when you are focused and free from distractions.

Energized work means taking steps to make sure you are able physically and mentally to get into a focused state. This means do not overwork yourself (or let others overwork you). It also means stay healthy, and show respect to your teammates to keep them healthy.

Pair Programming

Pair Programming means all production software is developed by two people sitting at the same machine. The idea behind this practice is that two brains and four eyes are better than one brain and two eyes. You effectively get a continuous code review and quicker response to nagging problems that may stop one person dead in their tracks.

Teams that have used pair programming have found that it improves quality and does not actually take twice as long because they are able to work through problems quicker and they stay more focused on the task at hand, thereby creating less code to accomplish the same thing.

Stories

Describe what the product should do in terms meaningful to customers and users.

These stories are intended to be short descriptions of things users want to be able to do with the product that can be used for planning and serve as reminders for more detailed conversations when the team gets around to realizing that particular story.

Weekly Cycle

The Weekly Cycle is synonymous to an iteration. In the case of XP, the team meets on the first day of the week to reflect on progress to date, the customer picks the stories they would like delivered in that week, and the team determines how they will approach those stories. The goal by the end of the week is to have running tested features that realize the selected stories.

The intent behind the time boxed delivery period is to produce something to show to the customer for feedback.

Quarterly Cycle

The Quarterly Cycle is synonymous to a release. The purpose is to keep the detailed work of each weekly cycle in context of the overall project. The customer lays out the overall plan for the team in terms of features desired within a particular quarter, which provides the team with a view of the forest while they are in the trees, and it also helps the customer to work with other stakeholders who may need some idea of when features will be available.

Remember when planning a quarterly cycle the information about any particular story is at a relatively high level, the order of story delivery within a Quarterly Cycle can change and the stories included in the Quarterly Cycle may change. If you are able to revisit the plan on a weekly basis following each weekly cycle, you can keep everyone informed as soon as those changes become apparent to keep surprises to a minimum.

Slack

The idea behind slack in XP terms is to add some low priority tasks or stories in your weekly and quarterly cycles that can be dropped if the team gets behind on more important tasks or stories. Put another way, account for the inherent variability in

estimates to make sure you leave yourself a good chance of meeting your forecasts.

Ten-Minute Build

The goal with the Ten-Minute Build is to automatically build the whole system and run all of the tests in ten minutes. The founders of XP suggested a 10 minute time frame because if a team has a build that takes longer than that, it is less likely to be run on a frequent basis, thus introducing longer time between errors.

This practice encourages your team to automate your build process so that you are more likely to do it on a regular basis and to use that automated build process to run all of your tests.

This practice supports the practice of Continuous Integration and is supported by the practice of Test First Development.

Continuous Integration

Continuous Integration is a practice where code changes are immediately tested when they are added to a larger code base. The benefit of this practice is you can catch and fix integration issues sooner.

Most teams dread the code integration step because of the inherent discovery of conflicts and issues that result. Most teams take the approach "If it hurts, avoid it as long as possible".

Practitioners of XP suggest "if it hurts, do it more often".

The reasoning behind that approach is that if you experience problems every time you integrate code, and it takes a while to find where the problems are, perhaps you should integrate more often so that if there are problems, they are much easier to find because there are fewer changes incorporated into the build.

This practice requires some extra discipline and is highly dependent on Ten Minute Build and Test First Development.

Test-First Programming

Instead of following the normal path of:

develop code -> write tests -> run tests

The practice of Test-First Programming follows the path of:

Write failing automated test -> Run failing test -> develop code to make test pass -> run test -> repeat

As with Continuous Integration, Test-First Programming reduces the feedback cycle for developers to identify and resolve issues, thereby decreasing the number of bugs that get introduced into production.

Incremental Design

The practice of Incremental Design suggests that you do a little bit of work up front to understand the proper breadth-wise perspective of the system design, and then dive into the details of a particular aspect of that design when you deliver specific features. This approach reduces the cost of changes and allows you to make design decisions when necessary based on the most current information available.

The practice of Refactoring was originally listed among the 12 core, but was incorporated into the practice of Incremental Design. Refactoring is an excellent practice to use to keep the design simple, and one of the most recommended uses of refactoring is to remove duplication of processes.

Roles

Although Extreme Programming specifies particular practices for your team to follow, it does not really establish specific roles for the people on your team.

Depending on which source you read, there is either no guidance, or there is a description of how roles typically found in more traditional projects behave on Extreme Programming projects. Here are four most common roles associated with Extreme Programming:

The Customer

The Customer role is responsible for making all of the business decisions regarding the project including:

- What should the system do (What features are included and what do they accomplish)?

- How do we know when the system is done (what are our acceptance criteria)?
- How much do we have to spend (what is the available funding, what is the business case)?
- What should we do next (in what order do we deliver these features)?

The XP Customer is expected to be actively engaged on the project and ideally becomes part of the team.

The XP Customer is assumed to be a single person, however experience has shown that one person cannot adequately provide all of the business related information about a project. Your team needs to make sure that you get a complete picture of the business perspective, but have some means of dealing with conflicts in that information so that you can get clear direction.

The Developer

Because XP does not have much need for role definition, everyone on the team (with the exception of the customer and a couple of secondary roles listed below) is labeled a developer. Developers are responsible for realizing the stories identified by the Customer. Because different projects require a different mix of skills, and because the XP method relies on a cross functional team providing the appropriate mix of skills, the creators of XP felt no need for further role definition.

The Tracker

Some teams may have a tracker as part of their team. This is often one of the developers who spends part of their time each week filling this extra role. The main purpose of this role is to keep track of relevant metrics that the team feels necessary to track their progress and to identify areas for improvement. Key metrics that your team may track include velocity, reasons for changes to velocity, amount of overtime worked, and passing and failing tests.

This is not a required role for your team, and is generally only established if your team determines a true need for keeping track of several metrics.

The Coach

If your team is just getting started applying XP, you may find it helpful to include a Coach on your team. This is usually an outside consultant or someone from elsewhere in your organization who has used XP before and is included in your team to help mentor the other team members on the XP Practices and to help your team maintain your self discipline.

The main value of the coach is that they have gone through it before and can help your team avoid mistakes that most new teams make.

Lifecycle

To describe XP in terms of a lifecycle it is probably most appropriate to revisit the concept of the Weekly Cycle and Quarterly Cycle.
First, start off by describing the desired results of the project by having customers define a set of stories. As these stories are being created, the team estimates the size of each story. This size estimate, along with relative benefit as estimated by the customer can provide an indication of relative value which the customer can use to determine priority of the stories.
If the team identifies some stories that they are unable to estimate because they don't understand all of the technical considerations involved, they can introduce a spike to do some focused research on that particular story or a common aspect of multiple stories. Spikes are short, time-boxed time frames set aside for the purposes of doing research on a particular aspect of the project. Spikes can occur before regular iterations start or alongside ongoing iterations.
Next, the entire team gets together to create a release plan that everyone feels is reasonable. This release plan is a first pass at what stories will be delivered in a particular quarter, or release. The stories delivered should be based on what value they provide and considerations about how various stories support each other.
Then the team launches into a series of weekly cycles. At the beginning of each weekly cycle, the team (including the customer) gets together to decide which stories will be realized during that week. The team then breaks those stories into tasks to be completed within that week.
At the end of the week, the team and customer review progress to date and the customer can decide whether the project should continue, or if sufficient value has been delivered.

Origins

XP was first used on the Chrysler Comprehensive Compensation (C3) program which was initiated in the mid 90's and switched to an XP project when Kent Beck was brought on to the project to improve the performance of the system. He wound up adding a couple of other folks, including Ron Jeffries to the team and changing the way the team approached development. This project helped to bring the XP methodology into focus and the several books written by people who were on the project helped spread knowledge about and adaptation of this approach.

Primary Contributions

XP's primary contribution to the software development world is an interdependent collection of engineering practices that teams can use to be more effective and produce higher quality code. Many teams adopting agile start by using a different framework and when they identify the need for more disciplined engineering practices they adopt several if not all of the engineering practices espoused by XP.

An additional, and equally important, contribution of XP is the focus on practice excellence. The method prescribes a small number of absolutely essential practices and encourages teams to perform those practices as good as they possibly can, almost to the extreme. This is where the name comes from. Not because the practices themselves are necessarily radical (although some consider some of them pretty far out) rather that teams continuously focus so intently on continuously improving their ability to perform those few practices.

Further Reading