extreme-programming

# Extreme Programming: What Is It And How Do You Use It?

January 20, 2017 | Andrew Powell-Morse | SDLC

Extreme Programming is a software development methodology designed to improve the quality of software and its ability to properly adapt to the changing needs of the customer or client. During the mid and late nineties, while working on the Chrysler Comprehensive Compensation System (C3) to help manage the company's payroll, software engineer Ken Beck first developed the Extreme Programming methodology. In October 1999, he published *Extreme Programming Explained*, detailing the entire method for others, and shortly thereafter the official website was launched as well.

Similar to other Agile Methods of development, Extreme Programming aims to provide iterative and frequent small releases throughout the project, allowing both team members and customers to examine and review the project's progress throughout the entire SDLC.

Throughout this article, we'll examine exactly what Extreme Programming is and how it works, from the values and principles that are behind it, to the rules and procedural best practices that are used to implement a new Extreme Programming project, so let's get started!

Some more specific takes on SDLC include:

| | | |
|---|---|---|
| Rapid Application Development | Test-Driven Development | Waterfall Model |
| Iterative Model | Software Development Life Cycle | Scaled Agile Framework |
| Agile Model | Scrum | Rational Unified Process |
| Big Bang Model | V-Model | Conceptual Model |
| Kaizen Model | Kanban Model | Spiral Model |

## Extreme Values

These five fundamental values provide the foundation on which the entirety of the Extreme Programming paradigm is built, allowing the people involved in the project to feel confident in the direction the project is taking and to understand their personal feedback and insight is as necessary and welcome as anyone else.

Simplicity: We will do what is needed and asked for, but no more. This will maximize the value created for the investment made to date. We will take small simple steps to our goal and mitigate failures as they happen. We will create something we are proud of and maintain it long term for reasonable costs.

Communication: Everyone is part of the team and we communicate face to face daily. We will work together on everything from requirements to code. We will create the best solution to our problem that we can together.

Feedback: We will take every iteration commitment seriously by delivering working software. We demonstrate our software early and often then listen carefully and make any changes needed. We will talk about the project and adapt our process to it, not the other way around.

Respect: Everyone gives and feels the respect they deserve as a valued team member. Everyone contributes value even if it's simply enthusiasm. Developers respect the expertise of the customers and vice versa. Management respects our right to accept responsibility and receive authority over our own work.

Courage: We will tell the truth about progress and estimates. We don't document excuses for failure because we plan to succeed. We don't fear anything because no one ever works alone. We will adapt to changes when ever they happen.

## Extreme Rules

Initially published by Don Wells in 1999, the proprietor of the Extreme Programming website, this set of Extreme Programming Rules were originally intended to help to counter the claims that Extreme Programming fails to support some of the prominent disciplines necessary for modern development.

**Planning**

- User stories are written.
- Release planning creates the release schedule.
- Make frequent small releases.
- The project is divided into iterations.
- Iteration planning starts each iteration.

**Managing**

- Give the team a dedicated open work space.
- Set a sustainable pace.
- A stand up meeting starts each day.
- The Project Velocity is measured.
- Move people around.
- Fix Extreme Programming when it breaks.

**Designing**

- Simplicity.
- Choose a system metaphor.
- Use CRC cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.

**Coding**

- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Set up a dedicated integration computer.
- Use collective ownership.

**Testing**

- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.

## Extreme Practices

Created using what were considered the best practices of software development at the time, these twelve Extreme Programming Best Practices detail the specific procedures that should be followed when implementing a project using Extreme Programming.

### Fine-scale feedback

**Pair programming**

In essence, pair programming means that two people work in tandem on the same system when developing any production code. By frequently rotating partners throughout the team, Extreme Programming promotes better communication and team-building.

**Planning game**

Often this takes the form of a meeting at a frequent and well-defined interval (every one or two weeks), where the majority of planning for the project takes place.

Within this procedure exists the Release Planning stage, where determinations are made regarding what is required for impending releases. Sections of Release Planning include:

- Exploration Phase: Story cards are used to detail the most valuable requirements from customers.
- Commitment Phase: Planning and commitments from the team are made to meet the needs of the next schedule release and get it out on time.
- Steering Phase: This allows for previously developed plans to be adjusted based on the evolving needs of the project, similar to many other Agile Model methodologies.

Following the Release Planning is also the Iteration Planning section, which consists of the same three sub-phases of its own, but with variants on their implementations:

- Exploration Phase: All project requirements are written down.
- Commitment Phase: Necessary tasks yet to be completed to meet the upcoming iteration release are assigned to developers and scheduled appropriately.
- Steering Phase: Development takes place and, upon completion, the resulting iteration is compared to the outlined story cards created at the start of the Planning procedure.

**Test-driven development**

While an entire article could be written about test-driven development, the concept is fairly well known among developers and effectively means that tests are generated for each and every requirement of the project, and *only then* is code developed that will successfully pass those tests.

**Whole team**

As with many other SDLC methods and practices, Extreme Programming promotes the inclusion of customers and clients throughout the entire process, using their feedback to help shape the project at all times.

### Continuous process

**Continuous integration**

Another common practice in modern development, the idea behind continuous integration is that all code developed across the entire team is merged into one common repository many times a day. This ensures that any issues with integration across the entire project are noticed and dealt with as soon as possible.

**Code refactoring**

Another very common practice, the idea behind code refactoring is simply to improve and redesign the structure of already existing code, without modifying its fundamental behavior. Simple examples of refactoring include fixing improperly names variables or methods, and reducing repeated code down to a single method or function.

**Small releases**

Very much in line with the practices of the Iterative Model, this concept ensures that the project will feature iterated, small releases on a frequent basis, allowing the customer as well, as all team members, to get a sense of how the project is developing.

## Shared understanding

**Coding standards**

The coding standard is simply a set of best practices within the code itself, such as formatting and style, which the entire team abides by throughout the life cycle of the project. This promotes better understanding and readability of the code not only for current members, but for future developers as well.

**Collective code ownership**

This practice allows for any developer across the team to change any section of the code, as necessary. While this practice may sound dangerous to some, it speeds up development time, and any potential issues can be quelled with proper unit testing.

**Simple design**

There's little reason to complicate things whenever a simpler option is available. This basic practice of keeping all components and code as simple as can be ensures that the entire team is always evaluating whether things could be done in an easier way.

**System metaphor**

Best thought of as part of the coding standards, the system metaphor is the idea that every person on the team should be able to look at the high-level code that is developed, and have a clear understanding of what functionality that code is performing.

## Programmer welfare

**Sustainable pace**

A key concept for better work-life balance with developers on an Extreme Programming project is the notion that nobody should be required to work in excess of the normal scheduled work week. Overtime is frowned upon, as is the concept of "crunch time", where developers are expected to work extreme hours near the end of a release to get everything completed on time.

**Share this:**

**Related**

Extreme Programming for Extreme Programmers
August 27, 2013
In "internet"

Agile Model: What Is It And How Do You Use It?
January 16, 2017
In "SDLC"

Waterfall Model: What Is It and When Should You Use It?
December 8, 2016
In "SDLC"

Try Airbrake For Free

Latest From Our Blog

**Announcing Single Sign-on for All Paid Airbrake Plans**

Recently Airbraked announced the availability of SAML Single Sign-on for large teams. Since that

Useful Links

Airbrake Docs
API Docs
Blog
Status Site

Contact Us

535 Mission Street, 14th floor, San Francisco, CA 94105

801 Barton Springs Rd, Austin, TX 78704

announcement, we have received a...

**With a Little Help From My Trends**

You may already be familiar with Airbrake email digests which give you a snapshot of what happened to your...

1-888-479-8323
sales@airbrake.io
facebook.com/airbrake.io
@airbrake

Airbrake © 2020