


Extreme Programming


An Agile Process



Chad Levesque
October 27th, 2009
ENGI7893 – Software Engineering



Extreme Programming

- Definition
- Origins
- Core Values
- Best Practices
- Benefits & Limitations





What is it?

- Extreme Programming, or XP, is a methodology.
 - A set of principles and practices that guide the rapid development of software.
 - A type of Agile Development, focused on collaborative, iterative development.
 - Named because it takes 12 well-known software “best practices” to their logical extremes.

Origins

- In the 1990's, the dot-com boom and the increasing importance of speed-to-market introduced a strain on the software industry.
- Many development teams were facing problems where traditional practices were failing while attempting to adapt to customers' increasingly rapid requirements changing.





Origins

In 1996, Kent Beck was brought in to Chrysler to work on the Chrysler Comprehensive Compensation payroll system.

Noting problems in the development process, Beck took it as an opportunity to experiment with a new methodology. He took a set of software practices and implemented them at “extreme” levels.

His experience on the C3 project lead to the publishing of “Extreme Programming Explained” in 1999.



The first time I was asked to lead a team, I asked them to do a little bit of the things I thought were sensible, like testing and reviews. The second time there was a lot more on the line. I thought, “Damn the torpedoes, at least this will make a good article.” [and] asked the team to crank up all the knobs to 10 on the things I thought were essential and leave out everything else.
—Kent Beck

Core Values

- **Communication.**
 - Team members work face-to-face daily. Everyone on the team is involved in all aspects of development.
- **Simplicity.**
 - Develop what is required and no more. Take the “You’re not gonna need it” approach.
- **Feedback.**
 - Software is demonstrated early and often with frequent iterations (every 2 weeks.) The project is adapted around the feedback.
- **Courage.**
 - Developers tell the truth about progress and estimates; they refactor often, removing obsolete code with no sentimental attachment.
- **Respect.**
 - Team members respect each other as developers, and everyone works to contribute value. Developers respect the expertise of the customer, and management respects the developers' knowledge and responsibility over their work.



Best Practices

- Extreme Programming describes 12 best practices, grouped into 4 categories. These take general “good software ideas” and make them **EXTREME!**^{#%^^!none}
- **Fine scale feedback:** Pair programming, planning game, test-driven development, whole team.
- **Continuous process:** Continuous integration, refactoring, small releases.
- **Shared understanding:** Coding standards, collective code ownership, simple design, system metaphor.
- **Programmer welfare:** Sustainable pace.



Fine Scale Feedback

Pair Programming

If code reviews are good, we'll review code all the time.

- In pair programming, all code is developed by two programmers working in tandem at the same machine.
- Typically, the programmer “driving” the computer is responsible for implementation details, while the “passenger” provides continuous code review, and focuses on how the code fits into the system as a whole.
- Programmers switch frequently throughout a work day, and pairs are rotated on a regular basis.
- Studies have shown that this method produces more defect-free code, better software designs, and increased morale over solitary programming. *



* Laurie Williams, University of Utah Computer Science, 2000.

Fine Scale Feedback

The Planning Game

If short iterations are good, we'll make the iterations really, really short.

- Iterations are very short: one or two weeks.
- Planning is done in two phases, Release and Iteration.
- Release Planning: Customers and developers work together to prioritize requirements for near-term releases. The customers will express their desires with User Stories.
- Iteration Planning: The developers translate requirements into task cards and assign a value to each based on risk or value. Programmers commit to a selection of task cards based on their performance in the last iteration.



Copyright © 2003 Shari Parton, Supt.com, Inc.

Fine Scale Feedback

Test Driven Development

If testing is good, everybody will test all the time.

- Unit tests are written before production code; production code is only written to make failing tests pass.
- No code can be checked in while tests are failing.

Whole Team

If communication with the customer is good, have the customer as part of the on-site team.

- Where possible, a representative of the software's end-user is frequently present for face-to-face consultations. Ideally, the customer is a full-time member of the team.

Continuous Process

Continuous Integration

If integration testing is important, then we'll integrate and test several times a day (continuous integration).

- Everyone works on the most recent version of the code. Code is committed to the repository every few hours.

Design Improvement

If design is good, we'll make it part of everybody's daily business.

- XP'ers are constantly refactoring their code to ensure the design stays as simple as possible, and code rot is avoided.

Small Releases

If frequent releases are good, we'll release all the time.

- Extreme Programming pushes for alpha builds to be done frequently, to demonstrate features to the customer as they are implemented.

Shared Understanding

Coding Standards

If consistency is important, we'll ensure that every developer follows the exact same coding standard.

- Developers strictly adhere a coding standard agreed upon before the project begins. This standard ensures consistent style and formatting and makes it easier for the team to share code.

Collective Code Ownership

If it's good to disseminate knowledge and share responsibility, involve everyone in everything.

- Everybody is responsible for all the code. Through pair programming, every developer on the team gets to work on every aspect of the system.

Shared Understanding

Simple Design

If simplicity is good, we'll keep the system as simple as possible at all times.

- The developers always implement the simplest design possible to support the current functionality, without worrying about anticipating change or future design problems.

System Metaphor

If architecture is important, everybody will always work at defining and refining the design.

- The system is described in terms of a metaphor, which helps guide the developers' mental models and provides an easy-to-understand way of communicating with the customer.

Programmer Welfare

Sustainable Pace

If overtime causes stress and reduces morale/productivity, we will always work at a sustainable pace.

- XP'ers are encouraged to work only 40-hours a week. Careful planning and honest time estimates should eliminate the need for overtime.



Benefits & Limitations

Benefits

- It has been empirically demonstrated that the adaptation of Extreme Programming values and practices leads to an improvement in productivity, quality, and morale over traditional software development methods.

Limitations

- It has yet to be proven that these methods scale appropriately with team size. It has only been found to work efficiently with teams of less than 25 or so.

References

- Kent Beck, *Extreme Programming Explained: Embrace Change* Addison-Wesley, 2000.
- R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.
- "Everyone's a Programmer" by Clair Tristram. *Technology Review*, Nov 2003. p. 39
- "eXtreme programming principles & practices" by Sylvia Ilieva, Eliza Stefanova, Sofia University
- <http://www.extremeprogramming.org> (Don Wells)
- <http://www.xp.co.nz/> (Ian Mitchell)