

EXTREME PROGRAMMING (XP)



Formulated in 1999 by Kent Beck, Ward Cunningham and Ron Jeffries

Agile software development methodology (others: Scrum, DSDM, Kanban)

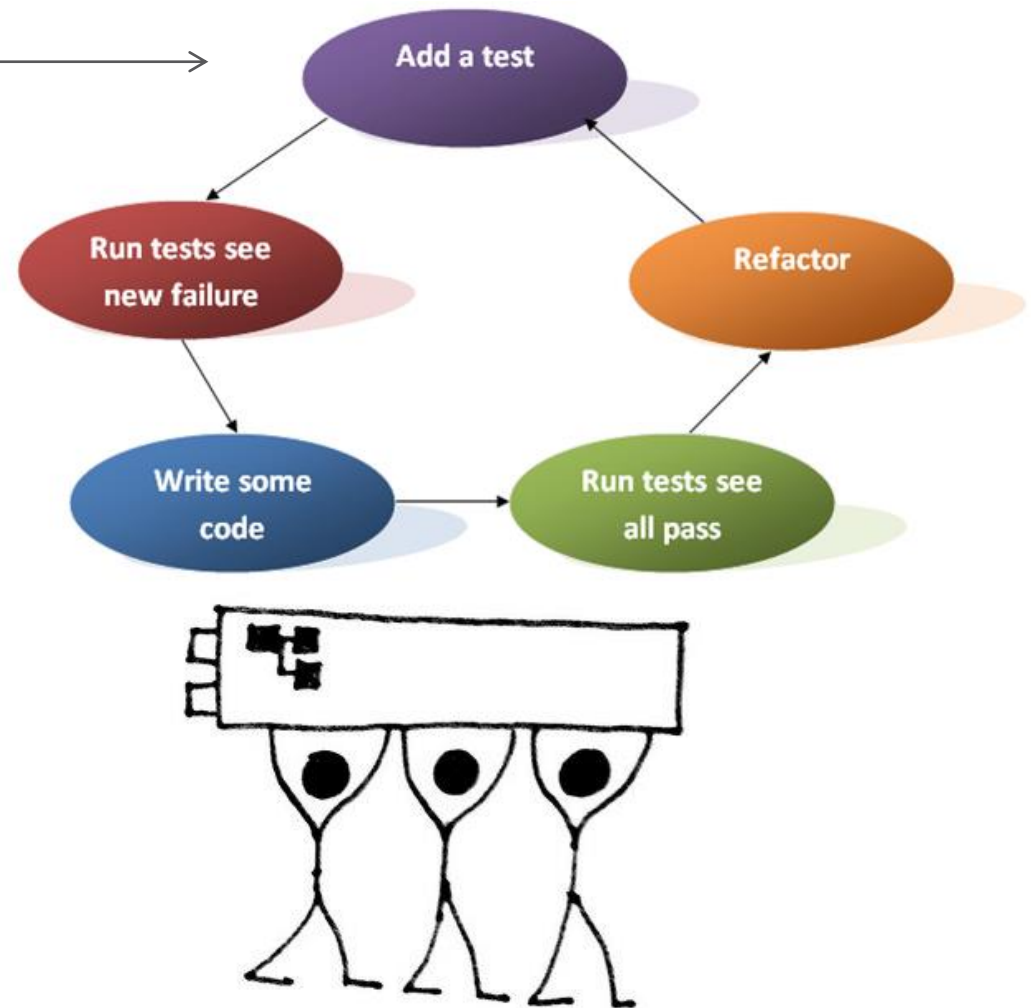
Developed in reaction to high ceremony methodologies

EXAMPLE OF PRINCIPLES FROM XP (EXTREME PROGRAMMING)

› Test Driven Development

› Continuous Integration

› Collective Code Ownership



XP: WHY?

Previously:

- Get all the requirements before starting design
- Freeze the requirements before starting development
- Resist changes: they will lengthen schedule
- Build a change control process to ensure that proposed changes are looked at carefully and no change is made without intense scrutiny
- Deliver a product that is obsolete on release

XP: EMBRACE CHANGE

Recognize that:

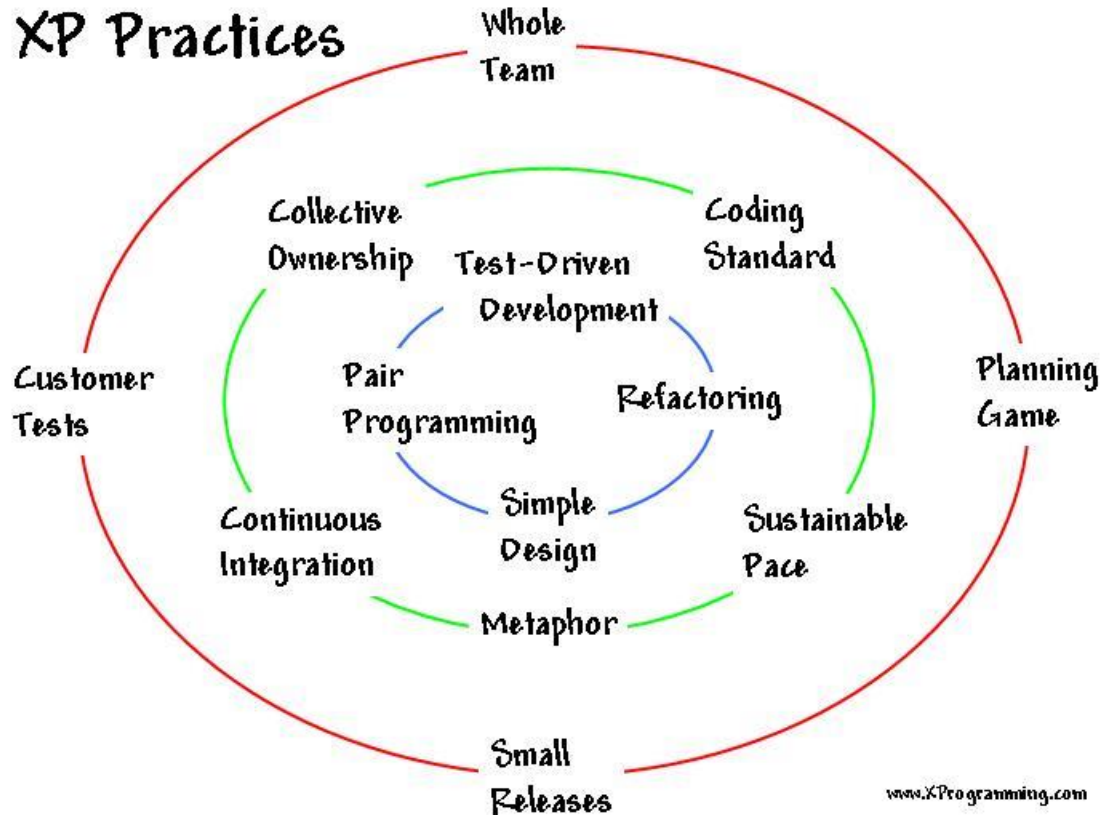
- All requirements will not be known at the beginning
- Requirements will change

Use tools to accommodate change as a natural process

Do the simplest thing that could possibly work and refactor mercilessly

Emphasize values and principles rather than process

XP PRACTICES



(Source: <http://www.xprogramming.com/xpmag/whatisxp.htm>)

THE XP TEAM

How to design and program the software

- programmers, designers, and architects

Where defects are likely to hide

- testers

Why the software is important

- product manager

The rules the software should follow

- domain experts

How the software should behave

- interaction designers

How the user interface should look

- graphic designers

How to interact with the rest of the company

- project manager

Where to improve work habits

- coach

XP PRACTICES: WHOLE TEAM

All contributors to an XP project are one team

Must include a business representative: the 'Customer'


- Provides requirements
- Sets priorities
- Steers project

Team members are programmers, testers, analysts, coach, manager

Best XP teams have no specialists

XP TEAM SIZE

Assume teams with 4 to 10 programmers (5 to 20 total team members).



Applying the staffing guidelines to a team of 6 programmers produces a team that also includes 4 customers, 1 tester, and a project manager, for a total team size of 12 people.

FULL-TIME TEAM MEMBERS

All the team members should sit with the team full-time and give the project their complete attention.



This particularly applies to customers, who are often surprised by the level of involvement XP requires of them.

XP PRACTICES: PLANNING GAME

Two key questions in software development:

- Predict what will be accomplished by the due date
- Determine what to do next

Need is to steer the project

Exact prediction (which is difficult) is not necessary

XP PRACTICES: PLANNING GAME

XP Release Planning

- Customer presents required features
- Programmers estimate difficulty
- Imprecise but revised regularly

XP Iteration Planning

- Two week iterations
- Customer presents features required
- Programmers break features down into tasks
- Team members sign up for tasks
- Running software at end of each iteration

XP PRACTICES: CUSTOMER TESTS

The Customer defines one or more automated acceptance tests for a feature

Team builds these tests to verify that a feature is implemented correctly

Once the test runs, the team ensures that it keeps running correctly thereafter

System always improves, never backslides

XP PRACTICES: SMALL RELEASES

Team releases running, tested software every iteration

Releases are small and functional

The Customer can evaluate or in turn, release to end users, and provide feedback

Important thing is that the software is visible and given to the Customer at the end of every iteration

XP PRACTICES: SIMPLE DESIGN

Build software to a simple design

Through programmer testing and design improvement, keep the software simple and the design suited to current functionality

Design steps in release planning and iteration planning

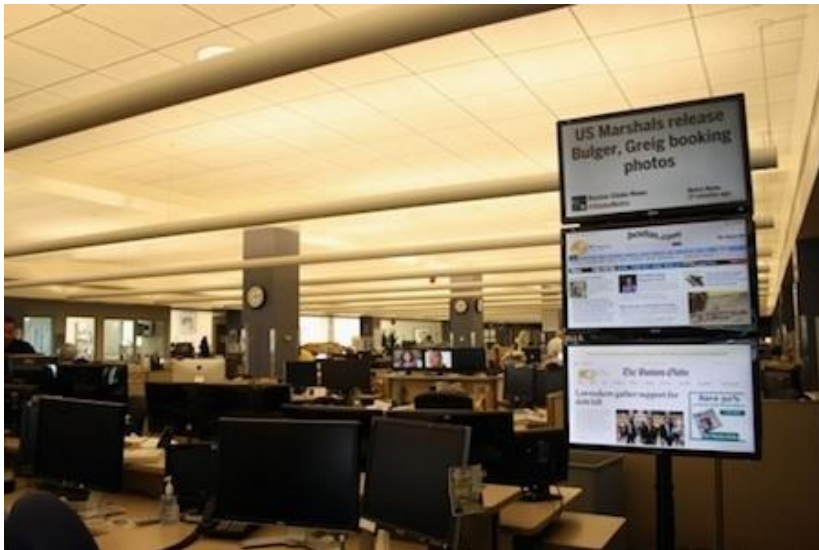
Teams design and revise design through refactoring, through the course of the project

XP PRACTICES: INFORMATIVE WORKSPACE

Your workspace is the cockpit of your development effort: create an informative workspace

An informative workspace broadcasts information into the room (eg. radiators)

It's improve stakeholder trust



XP PRACTICES: PAIR PROGRAMMING

All production software is built by two programmers, sitting side by side, at the same machine

All production code is therefore reviewed by at least one other programmer

Research into pair programming shows that pairing produces better code in the same time as programmers working singly

Pairing also communicates knowledge throughout the team

XP PRACTICES: TEST-DRIVEN DEVELOPMENT

Teams practice TDD by working in short cycles of adding a test, and then making it work

Easy to produce code with 100 percent test coverage

These programmer tests or unit tests are all collected together

Each time a pair releases code to the repository, every test must run correctly

XP PRACTICES: DESIGN IMPROVEMENT

Continuous design improvement process called 'refactoring':

- Removal of duplication
- Increase cohesion
- Reduce coupling

Refactoring is supported by comprehensive testing - customer tests and programmer tests

XP PRACTICES: CONTINUOUS INTEGRATION

Teams keep the system fully integrated at all times

Daily, or multiple times a day builds

Avoid 'integration hell'

Avoid code freezes

10 minutes build

XP PRACTICES: COLLECTIVE CODE OWNERSHIP

Any pair of programmers can improve any code at any time

All code gets the benefit of many people's attention

Avoid duplication

Programmer tests catch mistakes

Pair with expert when working on unfamiliar code

XP PRACTICES: CODING STANDARD

Use common coding standard



All code in the system must look as though written by an individual



Code must look familiar, to support collective code ownership

XP PRACTICES: SUSTAINABLE PACE

Team will produce high quality product when not overly exerted

Avoid overtime, maintain 40 hour weeks

'Death march' projects are unproductive and do not produce quality software

Work at a pace that can be sustained indefinitely

CHARACTERISTICS OF SUCCESSFUL XP PROJECTS



Very rapid development

Exceptional responsiveness to user and customer change requests

High customer satisfaction

Amazingly low error rates

System begins returning value to customers very early in the process

XP VALUES

Communication

Simplicity

Feedback

Courage

XP VALUES: COMMUNICATION

Poor communication in software teams is one of the root causes of failure of a project

Stress on good communication between all stakeholders-- customers, team members, project managers

Customer representative always on site

Paired programming

XP VALUES: SIMPLICITY

‘Do the Simplest Thing That Could Possibly Work’

- Implement a new capability in the simplest possible way
- Refactor the system to be the simplest possible code with the current feature set

‘You Aren’t Going to Need It’ (YAGNI)

- Never implement a feature you don’t need now

YOU AREN'T GONNA NEED IT (YAGNI)

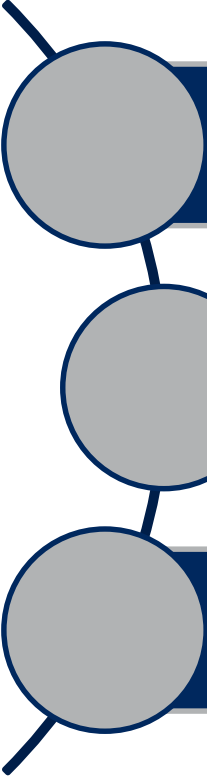
Important aspect of simple design: avoid speculative coding.

- Whenever you're tempted to add something to your design, ask yourself if it supports the stories and features you're currently delivering. If not, well... **you aren't gonna need it**. Your design could change. Your customers' minds could change.

Similarly, remove code that's no longer in use.

- You'll make the design smaller, simpler, and easier to understand. If you need it again in the future, you can always get it out of version control. For now, it's a maintenance burden you don't need.

XP VALUES: FEEDBACK



Always a running system that delivers information about itself in a reliable way

The system and the code provides feedback on the state of development

Catalyst for change and an indicator of progress

XP VALUES: COURAGE

Projects are
people-centric

Ingenuity of people
and not any
process that
causes a project to
succeed

XP CRITICISM

Unrealistic--
programmer
centric, not
business focused

Detailed
specifications are
not written

Design after
testing

Constant
refactoring

Customer
availability

12 practices are
too
interdependent

XP THOUGHTS

The best design is the code.

Testing is good. Write tests before code. Code is complete when it passes tests.

Simple code is better. Write only code that is needed. Reduce complexity and duplication.

Keep code simple. Refactor.

Keep iterations short. Constant feedback.

COMMON XP MISCONCEPTIONS

No written design documentation

- *Truth: no formal standards for how much or what kind of docs are needed.*

No design

- *Truth: minimal explicit, up-front design; design is an explicit part of every activity through every day.*

XP is easy

- *Truth: although XP does try to work with the natural tendencies of developers, it requires great discipline and consistency.*

MORE MISCONCEPTIONS

XP is just legitimized hacking

- *Truth: XP has extremely high quality standards throughout the process*
- *Unfortunate truth: XP is sometimes **used as an excuse** for sloppy development*

XP is the one, true way to build software

- *Truth: it seems to be a sweet spot for certain kinds of projects*

XP SUMMARY (BY ISTQB)

Values:

- communication, simplicity, feedback, courage, respect

Principles:

- humanity, economics, mutual benefit, self-similarity, improvement, diversity, reflection, flow, opportunity, redundancy, failure, quality, baby steps, accepted responsibility

Primary practices:

- sit together, whole team, informative workspace (radiators), energized work, pair programming, stories, weekly cycle, quarterly cycle, slack (do not use 100% allocation), 10 minute build, continuous integration, test first programming, incremental design

Many other agile practices use some aspects of XP

HOW IT ALL FITS TOGETHER

