



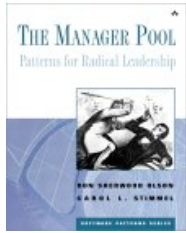
How good do you want to be, and when?
Let Ron Jeffries, experienced author, trainer, practitioner, coach, help your Agile team get to the next level! Bring your project in on time, on budget, fit for purpose!

search

RSS Feed

Sections:

- Welcome
- What is XP?
- Articles
- Software
- Community
- Archives



The Manager Pool
Don Sherwood Olson
and Carol L. Stimmel



Ruby in a Nutshell
Yukihiro Matsumoto

Xprogramming » What is Extreme Programming?

COLLECTED TOPICS: [Kate Oneal](#) | [Adventures in C#](#) | [Documentation in XP](#) | [Book Reviews](#)

What is Extreme Programming?

Ron Jeffries 11/08/2001

Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.



In Extreme Programming, every contributor to the project is an integral part of the "**Whole Team**". The team forms around a business representative called "the Customer", who sits with the team and works with them daily.

- Core Practices: [Whole Team](#)

Extreme Programming teams use a simple form of planning and tracking to decide what should be done next and to predict when the project will be done. Focused on business value, the team produces the software in a series of small fully-integrated releases that pass all the tests the Customer has defined.

- Core Practices: [Planning Game](#), [Small Releases](#), [Customer Tests](#)

Extreme Programmers work together in pairs and as a group.

Ron's books & publications



Extreme Programming Adventures in C#
Ron Jeffries

Calendar


2009 outlook:
Pushing on
The Nature of
Software
Development

2009 Training

Certified Scrum Plus!
Learn how to get
Things done!

2009 Conferences:
Scrum Gathering
Agile 2009
Simple Design
... stay tuned

amazon.com
and you're done.



[Lean Software Development](#)
Mary Poppendieck, ...
New \$46.15
Best \$22.15

[Agile Testing](#)
Lisa Crispin, Jane...

[User Stories Applied](#)
Mike Cohn

[Agile Principles, Patterns, and Prac...](#)
Robert C. Martin, ...

[Clean Code](#)
Robert C. Martin

[Privacy Information](#)

with simple design and obsessively tested code, improving the design continually to keep it always just right for the current needs.

- Core Practices: [Simple Design](#), [Pair Programming](#), [Test-Driven Development](#), [Design Improvement](#)

The Extreme Programming team keeps the system integrated and running all the time. The programmers write all production code in pairs, and all work together all the time. They code in a consistent style so that everyone can understand and improve all the code as needed.

- Core Practices: [Continuous Integration](#), [Collective Code Ownership](#), [Coding Standard](#)

The Extreme Programming team shares a common and simple picture of what the system looks like. Everyone works at a pace that can be sustained indefinitely.

- Core Practices: [Metaphor](#), [Sustainable Pace](#)

Core Practices

All the contributors to an XP project sit together, members of one team. This team must include a business representative — the “Customer” — who provides the requirements, sets the priorities, and steers the project. It’s best if the Customer or one of her aides is a real end user who knows the domain and what is needed. The team will of course have programmers. The team may include testers, who help the Customer define the customer acceptance tests. Analysts may serve as helpers to the Customer, helping to define the requirements. There is commonly a coach, who helps the team keep on track, and facilitates the process. There may be a manager, providing resources, handling external communication, coordinating activities. None of these roles is necessarily the exclusive property of just one individual: Everyone on an XP team contributes in any way that they can. The best teams have no specialists, only general contributors with special skills.

XP planning addresses two key questions in software development: predicting what will be accomplished by the due date, and determining what to do next. The emphasis is on steering the project — which is quite straightforward — rather than on exact prediction of what will be needed and how long it will take — which is quite difficult. There are two key planning steps in XP, addressing these two questions:

Release Planning is a practice where the Customer presents the desired features to the programmers, and the programmers estimate their difficulty. With the cost estimates in hand, and with knowledge of the importance of the features, the Customer lays out a plan for the project. Initial release plans are necessarily imprecise: neither the priorities nor the estimates are truly solid, and until the team begins to work, we won’t know just how fast they will go. Even the first release plan is accurate enough for decision making, however, and XP teams revise the release plan regularly.

Iteration Planning is the practice whereby the team is given direction every couple of weeks. XP teams build software in two-week “iterations”, delivering running useful software at the

end of each iteration. During Iteration Planning, the Customer presents the features desired for the next two weeks. The programmers break them down into tasks, and estimate their cost (at a finer level of detail than in Release Planning). Based on the amount of work accomplished in the previous iteration, the team signs up for what will be undertaken in the current iteration.

These planning steps are very simple, yet they provide very good information and excellent steering control in the hands of the Customer. Every couple of weeks, the amount of progress is entirely visible. There is no “ninety percent done” in XP: a feature story was completed, or it was not. This focus on visibility results in a nice little paradox: on the one hand, with so much visibility, the Customer is in a position to cancel the project if progress is not sufficient. On the other hand, progress is so visible, and the ability to decide what will be done next is so complete, that XP projects tend to deliver more of what is needed, with less pressure and stress.

As part of presenting each desired feature, the XP Customer defines one or more automated acceptance tests to show that the feature is working. The team builds these tests and uses them to prove to themselves, and to the customer, that the feature is implemented correctly. Automation is important because in the press of time, manual tests are skipped. That’s like turning off your lights when the night gets darkest.

The best XP teams treat their customer tests the same way they do **programmer tests**: once the test runs, the team keeps it running correctly thereafter. This means that the system only improves, always notching forward, never backsliding.

XP teams practice small releases in two important ways:

First, the team releases running, tested software, delivering business value chosen by the Customer, every iteration. The Customer can use this software for any purpose, whether evaluation or even release to end users (highly recommended). The most important aspect is that the software is visible, and given to the customer, at the end of every iteration. This keeps everything open and tangible.

Second, XP teams release to their end users frequently as well. XP Web projects release as often as daily, in house projects monthly or more frequently. Even shrink-wrapped products are shipped as often as quarterly.

It may seem impossible to create good versions this often, but XP teams all over are doing it all the time. See **Continuous Integration** for more on this, and note that these frequent releases are kept reliable by XP’s obsession with testing, as described here in **Customer Tests** and **Test-Driven Development**.

XP teams build software to a simple design. They start simple, and through **programmer testing** and **design improvement**, they keep it that way. An XP team keeps the design exactly suited for the current functionality of the system. There is no wasted motion, and the software is always ready for what’s next.

Design in XP is not a one-time thing, or an up-front thing, it is an all-the-time thing. There are design steps in release planning and iteration planning, plus teams engage in quick design sessions and design revisions through refactoring, through the course of the entire project. In an incremental,

iterative process like Extreme Programming, good design is essential. That's why there is so much focus on design throughout the course of the entire development.

All production software in XP is built by two programmers, sitting side by side, at the same machine. This practice ensures that all production code is reviewed by at least one other programmer, and results in better design, better testing, and better code.

It may seem inefficient to have two programmers doing "one programmer's job", but the reverse is true. [Research into pair programming](#) shows that pairing produces better code in about the same time as programmers working singly. That's right: two heads really are better than one!

Some programmers object to pair programming without ever trying it. It does take some practice to do well, and you need to do it well for a few weeks to see the results. Ninety percent of programmers who learn pair programming prefer it, so we highly recommend it to all teams.

Pairing, in addition to providing better code and tests, also serves to communicate knowledge throughout the team. As pairs switch, everyone gets the benefits of everyone's specialized knowledge. Programmers learn, their skills improve, they become more valuable to the team and to the company. Pairing, even on its own outside of XP, is a big win for everyone.

Extreme Programming is obsessed with feedback, and in software development, good feedback requires good testing. Top XP teams practice "test-driven development", working in very short cycles of adding a test, then making it work. Almost effortlessly, teams produce code with nearly 100 percent test coverage, which is a great step forward in most shops. (If your programmers are already doing even more sophisticated testing, more power to you. Keep it up, it can only help!)

It isn't enough to write tests: you have to run them. Here, too, Extreme Programming is extreme. These "programmer tests", or "unit tests" are all collected together, and every time any programmer releases any code to the repository (and pairs typically release twice a day or more), every single one of the programmer tests must run correctly. One hundred percent, all the time! This means that programmers get immediate feedback on how they're doing. Additionally, these tests provide invaluable support as the software design is improved.

Extreme Programming focuses on delivering business value in every iteration. To accomplish this over the course of the whole project, the software must be well-designed. The alternative would be to slow down and ultimately get stuck. So XP uses a process of continuous design improvement called *Refactoring*, from the title of Martin Fowler's book, "[Refactoring: Improving the Design of Existing Code](#)".

The refactoring process focuses on removal of duplication (a sure sign of poor design), and on increasing the "cohesion" of the code, while lowering the "coupling". High cohesion and low coupling have been recognized as the hallmarks of well-designed code for at least thirty years. The result is that XP teams start with a good, simple design, and always have a good, simple design for the software. This lets them sustain their development speed, and in fact generally increase speed as the project goes forward.

Refactoring is, of course, strongly supported by comprehensive testing to be sure that as the design evolves, nothing is broken. Thus the **customer tests** and **programmer tests** are a critical enabling factor. The XP practices support each other: they are stronger together than separately.

Extreme Programming teams keep the system fully integrated at all times. We say that daily builds are for wimps: XP teams build multiple times per day. (One XP team of forty people builds at least eight or ten times per day!)

The benefit of this practice can be seen by thinking back on projects you may have heard about (or even been a part of) where the build process was weekly or less frequently, and usually led to “integration hell”, where everything broke and no one knew why.

Infrequent integration leads to serious problems on a software project. First of all, although integration is critical to shipping good working code, the team is not practiced at it, and often it is delegated to people who are not familiar with the whole system. Second, infrequently integrated code is often — I would say usually — buggy code. Problems creep in at integration time that are not detected by any of the testing that takes place on an unintegrated system. Third, weak integration process leads to long code freezes. Code freezes mean that you have long time periods when the programmers could be working on important shippable features, but that those features must be held back. This weakens your position in the market, or with your end users.

On an Extreme Programming project, any pair of programmers can improve any code at any time. This means that all code gets the benefit of many people’s attention, which increases code quality and reduces defects. There is another important benefit as well: when code is owned by individuals, required features are often put in the wrong place, as one programmer discovers that he needs a feature somewhere in code that he does not own. The owner is too busy to do it, so the programmer puts the feature in his own code, where it does not belong. This leads to ugly, hard-to-maintain code, full of duplication and with low (bad) cohesion.

Collective ownership could be a problem if people worked blindly on code they did not understand. XP avoids these problems through two key techniques: the **programmer tests** catch mistakes, and **pair programming** means that the best way to work on unfamiliar code is to pair with the expert. In addition to ensuring good modifications when needed, this practice spreads knowledge throughout the team.

XP teams follow a common coding standard, so that all the code in the system looks as if it was written by a single — very competent — individual. The specifics of the standard are not important: what is important is that all the code looks familiar, in support of collective ownership.

Extreme Programming teams develop a common vision of how the program works, which we call the “metaphor”. At its best, the metaphor is a simple evocative description of how the program works, such as “this program works like a hive of bees, going out for pollen and bringing it back to the hive” as a description for an agent-based information retrieval system.

Sometimes a sufficiently poetic metaphor does not arise. In any case, with or without vivid imagery, XP teams use a common system of names to be sure that everyone

understands how the system works and where to look to find the functionality you're looking for, or to find the right place to put the functionality you're about to add.

Extreme Programming teams are in it for the long term. They work hard, and at a pace that can be sustained indefinitely. This means that they work overtime when it is effective, and that they normally work in such a way as to maximize productivity week in and week out. It's pretty well understood these days that death march projects are neither productive nor produce quality software. XP teams are in it to win, not to die.

Conclusion

Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.

Picture



Here's a [picture](#) showing the practices and the main "cycles" of XP.