PRACTICING XP-THINKING

Thinking Practices of XP

Sometimes the biggest gains in productivity come from stopping to think about what you're doing, why you're doing it, and whether it's a good idea. The best developers don't just find something that works and use it; they also question why it works, try to understand it, and then improve it.

XP doesn't require experts. It does require a habit of mindfulness.

Five practices to help mindful developers

- **Pair programming** doubles the brainpower available during coding, and gives one person in each pair the opportunity to think about strategic, long-term issues.
- **Energized work** acknowledges that developers do their best, most productive work when they're energized and motivated.
- An **Informative workspace** gives the whole team more opportunities to notice what's working well and what isn't.
- **Root-cause analysis** is a useful tool for identifying the underlying causes of your problems.
- **Retrospectives** provide a way to analyze and improve the entire development process.

Pair Programming

Audience: Programmers, Whole Team

In which two programmers work together at one workstation. One, the driver, writes code while the other, the observer or navigator, reviews each line of code as it is typed in.

“We help each other succeed”

Pair programming is one of the first things people notice about XP. As this is about thinking practices to help mindful developers, it includes pair programming as the first practice. That’s because pair programming is all about increasing your brainpower.

Benefits of pair programming

- Fewer coding mistakes
- Knowledge is spread among the pairs
- Reduced effort to coordinate
- Increased resiliency(recover quickly from difficulties)

Challenges of pair programming

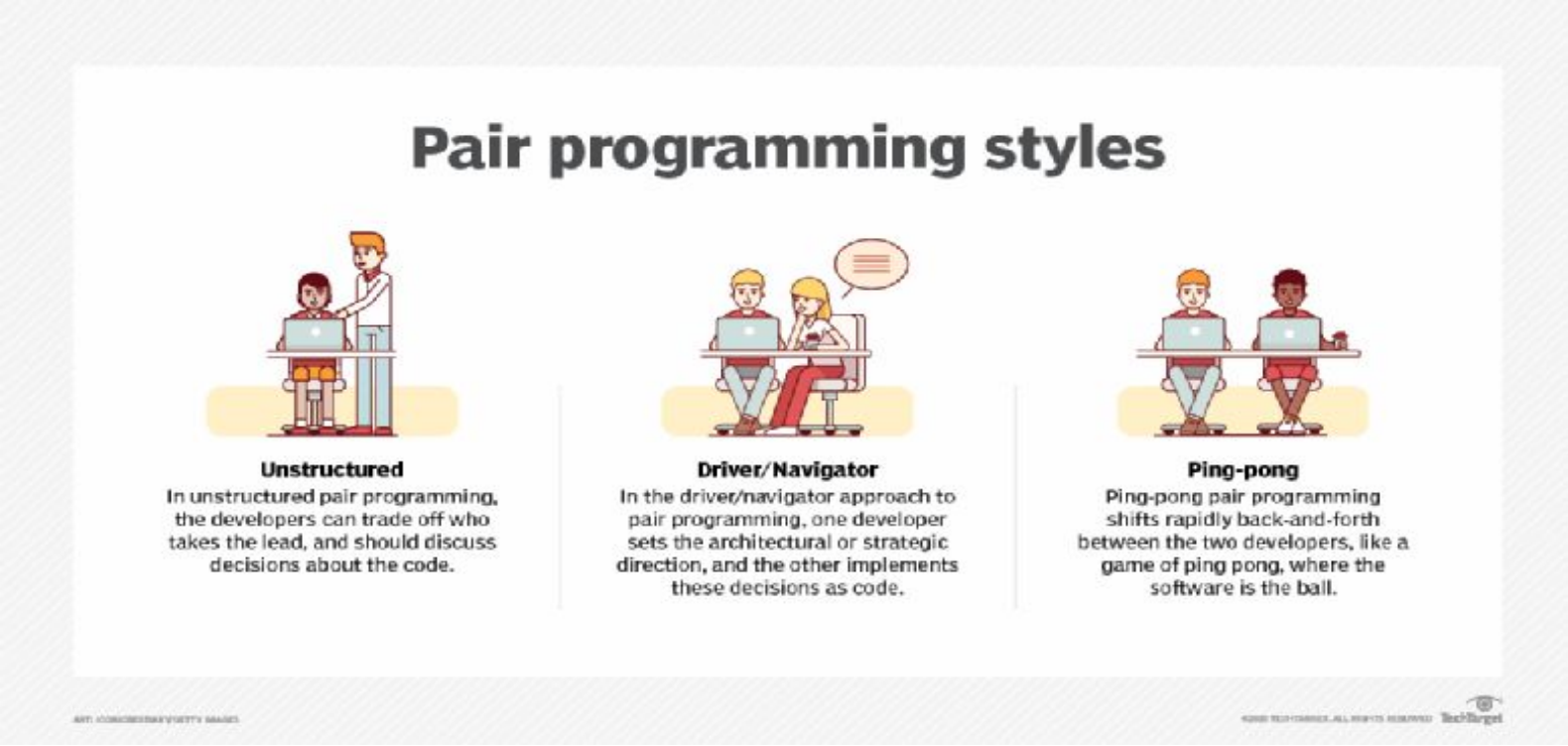
- Efficiency
- Equally engaging pairs
- Social and interactive process
- Sustainability

Pair programming styles and techniques

- Driver/navigator style
- Unstructured style
- Ping-pong style

Potential skill pairs

- Expert/expert pairs
- Novice/novice pairs
- Expert/novice pairs



Best practices for pair programming

- Consistent communication
- Switch roles consistently
- Pair up carefully
- Use a familiar development environment
- Submit code frequently
- Ask for clarification when needed
- Take breaks when needed

Results

When you pair program well, you find yourself focusing intently on the code and on your work with your partner.

The team as a whole enjoys higher quality code. Technical debt decreases. Knowledge travels quickly through the team, raising everyone’s level of competence and helping to integrate new team members quickly.

Contraindications

- If your workspace doesn't allow programmers to sit side by side comfortably, either change the workspace or don't pair program.
- If your team doesn't sit together, pairing may not work for you.
- Programmer resistance may be another reason to avoid pairing.

Alternatives

Pairing is a very powerful tool. It reduces defects, improves design quality, shares knowledge amongst team members, supports self-discipline, and reduces distractions, all without sacrificing productivity. If you cannot pair program, you need alternatives.

Formal code inspections can reduce defects, improve quality, and support self-discipline.

Inspections alone are unlikely to share knowledge as thoroughly as collective code ownership requires. If you cannot pair program, consider avoiding collective ownership, at least at first.

Energized Work

Audience: Coaches, Whole Team

XP's practice of energized work recognizes that, although professionals can do good work under difficult circumstances, they do their best, most productive work when they're energized and motivated..

“We work at a pace that allows us to do our best, most productive work indefinitely.”

How to Be Energized

“Go home on time”

- One of the simplest ways to be energized is to take care of yourself.
- While at work, give it your full attention.
- Energized work requires a supportive workplace and home life.

Supporting Energized Work

“Stay home when you're sick. You risk getting other people sick, too.“

- Tired people make mistakes and take shortcuts. The resulting errors can end up costing more than the work is worth.
- Pair programming is another way to encourage energized work.
- Creating and communicating this vision is the product manager's responsibility.
- The Planning Game supports energized work

Taking Breaks

“Stop when you're making more mistakes than progress.”

- When you're making more mistakes than progress, it's time to take a break.
- For programmers, switching pairs can help.
- In a highly collaborative environment, going dark—not talking—can also be a sign that someone needs a break.

Results

When your team is energized, there's a sense of excitement and camaraderie. As a group, you pay attention to detail and look for opportunities to improve your work habits. You make consistent progress every week and feel able to maintain that progress indefinitely. You value health over short-term progress and feel productive and successful.

Contraindications

Energized work is not an excuse to goof off. Generate trust by putting in a fair day's work.

Some organizations may make energized work difficult. If your organization uses the number of hours worked as a yardstick to judge dedication, you may be better off sacrificing energized work and working long hours. The choice between quality of life and career advancement is a personal one that only you and your family can make.

Alternatives

If your organization makes energized work difficult, mistakes are more likely. Pair programming can help tired programmers stay focused and catch each other's errors. Additional testing may be necessary to find the extra defects. If you can, add additional contingency time to your release plan for fixing them.

Informative Workspace

Audience: Whole Team

“We are tuned in to the status of our project.”

- Workspace is the cockpit of your development effort.
- Arrange your workspace with information necessary to steer your project: create an informative workspace.
- An informative workspace also allows people to sense the state of the project just by walking into the room. It conveys status information without interrupting team members and helps improve stakeholder trust.

Subtle Cues

“Simply poking your head into a project room should give you information about the project.”

- The essence of an informative workspace is information. One simple source of information is the feel of the room.
- Besides the feel of the room, other cues communicate useful information quickly and subconsciously.
- A collaborative design sketch on a whiteboard can often communicate an idea far more quickly and effectively than a half-hour PowerPoint presentation.

Big Visible Charts

- An essential aspect of an informative workspace is the big visible chart. The goal of a big visible chart is to display information so simply and unambiguously that it communicates even from across the room.
- Another useful status chart is a team calendar, which shows important dates, iteration numbers, and when team members will be out of the office.

Hand-Drawn Charts

The benefits of the informative workspace stem from the information being constantly visible from everywhere in the room.

“Don’t rush to computerize.”

Creating or modifying a chart is as simple as drawing with pen and paper.

Process Improvement Charts

- One type of big visible chart measures specific issues that the team wants to improve.
- Create process improvement charts as a team decision, and maintain them as a team responsibility.

As an example, XP teams have successfully used charts to help improve:

- Amount of pairing
- Pair switching
- Build performance
- Support responsiveness
- Needless interruptions

Results

When you have an informative workspace, you have up-to-the-minute information about all the important issues your team is facing. You know exactly how far you've come and how far you have to go in your current plan, you know whether the team is progressing well or having difficulty, and you know how well you're solving problems.

Contraindications

If your team doesn’t sit together in a shared workspace, you probably won’t be able to create an effective informative workspace.

Alternatives

If your team doesn’t sit together, but has adjacent cubicles or offices, you might be able to achieve some of the benefits of an informative workspace by posting information in the halls or a common area. Teams that are more widely distributed may use electronic tools supplemented with daily stand-up meetings.

A traditional alternative is the weekly status meeting, but I find these dreary wastes of time that delay and confuse important information.

Root-Cause Analysis

Audience: Whole Team

“We prevent mistakes by fixing our process.”

Rather than blaming people, I blame the process. What is it about the way we work that allowed this mistake to happen? How can we change the way we work so that it’s harder for something to go wrong?

How to Find the Root Cause

A classic approach to root-cause analysis is to ask “why” five times.

Here’s a real-world example.

Problem: When we start working on a new task, we spend a lot of time getting the code into a working state.

How to Fix the Root Cause

- Root-cause analysis is a technique you can use for every problem you encounter, from the trivial to the significant. You can ask yourself “why” at any time. You can even fix some problems just by improving your own work habits.
- More often, however, fixing root causes requires other people to cooperate. If your team has control over the root cause, gather the team members, share your thoughts, and ask for their help in solving the problem.

When Not to Fix the Root Cause

“A mistake-proof process is neither achievable nor desirable.”

Over time, work will go more smoothly. Mistakes will become less severe and less frequent. Eventually—it can take months or years—mistakes will be notably rare.

At this point, you may face the temptation to over-apply root-cause analysis. Beware of thinking that you can prevent all possible mistakes. Fixing a root cause may add overhead to the process.

Results

When root-cause analysis is an instinctive reaction, your team values fixing problems rather than placing blame. Your first reaction to a problem is to ask how it could have possibly happened. Rather than feeling threatened by problems and trying to hide them, you raise them publicly and work to solve them.

Contraindications

The primary danger of root-cause analysis is that, ultimately, every problem has a cause outside of your control.

Don’t use this as an excuse not to take action. If a root cause is beyond your control, work with someone (such as your project manager) who has experience coordinating with other groups. In the meantime, solve the intermediate problems. Focus on what is in your control.

Although few organizations actively discourage root-cause analysis, you may find that it is socially unacceptable. If your efforts are called “disruptive” or a “waste of time,” you may be better off avoiding root-cause analysis.

Alternatives

You can always perform root-cause analysis in the privacy of your thoughts. You'll probably find that a lot of causes are beyond your control. Try to channel your frustration into energy for fixing processes that you can influence.

Retrospectives

Audience: Whole Team

“We continually improve our work habits.”

No process is perfect. Your team is unique, as are the situations you encounter, and they change all the time. You must continually update your process to match your changing situations. Retrospectives are a great tool for doing so.

Types of Retrospectives

The most common retrospective, the iteration retrospective, occurs at the end of every iteration.

In addition to iteration retrospectives, you can also conduct longer, more intensive retrospectives at crucial milestones. These release retrospectives, project retrospectives, and surprise retrospectives (conducted when an unexpected event changes your situation) give you a chance to reflect more deeply on your experiences and condense key lessons to share with the rest of the organization.

How to Conduct an Iteration Retrospective

Anybody can facilitate an iteration retrospective if the team gets along well. An experienced, neutral facilitator is best to start with.

Let events follow their natural pace:

1. Norm Kerth's Prime Directive

2. Brainstorming (30 minutes)

3. Mute Mapping (10 minutes)

4. Retrospective objective (20 minutes)

After you've acclimated to this format, change it. The retrospective is a great venue for trying new ideas.

The Prime Directive

“Never use a retrospective to place blame or attack individuals.”

The retrospective is an opportunity to learn and improve. The team should never use the retrospective to place blame or attack individuals.

Brainstorming

If everyone agrees to the Prime Directive, hand out index cards and pencils, then write the following headings on the whiteboard:

- Enjoyable
- Frustrating
- Puzzling
- Same
- More
- Less

Mute Mapping

Mute mapping is a variant of affinity mapping in which no one speaks. It's a great way to categorize a lot of ideas quickly.

You need plenty of space for this. Invite everyone to stand up, go over to the whiteboard, and slide cards

around. There are three rules:

1. Put related cards close together.

2. Put unrelated cards far apart.

3. No talking.

Retrospective Objective

"Frustrated that your favourite category lost? Wait a month or two. If it's important, it will win eventually."

- After the voting ends, one category should be the clear winner. If not, don't spend too much time; flip a coin or something.
- Discard the cards from the other categories. If someone wants to take a card to work on individually, that's fine, but not necessary. Remember, you'll do another retrospective next week. Important issues will recur.
- The retrospective objective is the goal that the whole team will work toward during the next iteration.

After the Retrospective

The retrospective serves two purposes: sharing ideas gives the team a chance to grow closer, and coming up with a specific solution gives the team a chance to improve.

Results

When your team conducts retrospectives well, your ability to develop and deliver software steadily improves. The whole team grows closer and more cohesive, and each group has more respect for the issues other groups face. You are honest and open about your successes and failures and are more comfortable with change.

Contraindications

The biggest danger in a retrospective is that it will become a venue for acrimony rather than for constructive problem solving. A skilled facilitator can help prevent this, but you probably don’t have such a facilitator on hand. Be very cautious about conducting retrospectives if some team members tend to lash out, attack, or blame others.

If only one or two team members are disruptive, and attempts to work the problem through with them are ineffective, you may be better off removing them from the team. Their antisocial influence probably extends beyond the retrospective, hurting teamwork and productivity.

Alternatives

Some organizations define organization-wide processes. Others assign responsibility for the process to a project manager, technical lead, or architect. Although these approaches might lead to a good initial process, they don’t usually lead to continuous process improvement, and neither approach fosters team cohesiveness.