PRACTICING XP-COLLABORATING

Collaborating Practices of XP

The purpose of collaborating is to explore the flow of information in your project.

Communication in the real world is a lot messier than it is in my image. There are no glowing lines to sterilely transport information from one brain to another. Instead, people have to work together. They have to ask questions, discuss ideas, and even disagree.

Practices to help team and its stakeholders collaborate efficiently and effectively

- **Sitting together** leads to fast, accurate communication.
- **Real customer involvement** helps the team understand what to build.
- A **ubiquitous language** helps team members understand each other.
- **Stand-up meetings** keep team members informed.
- **Coding standards** provide a template for seamlessly joining the team’s work together.
- **Iteration demos** keep the team’s efforts aligned with stakeholder goals.
- **Reporting** helps reassure the organization that the team is working well.

Sit Together

Audience: Whole Team, Coaches

“We communicate rapidly and accurately.”

Compared to an in-person discussion, teleconferences are slow and stuttered, with uncomfortable gaps in the conversation and people talking over each other.

What you may not have realized is how much this affects your work.

Accommodating Poor Communication

As the distance between people grows, the effectiveness of their communication decreases. Misunderstandings occur and delays creep in. People start guessing to avoid the hassle of waiting for answers. Mistakes appear.

To combat this problem, most development methods attempt to reduce the need for direct communication. It’s a sensible response. If questions lead to delays and errors, reduce the need to ask questions!

The primary tools teams use to reduce reliance on direct communication are development phases and work-in-progress documents.

A Better Way

In XP, the whole team—including experts in business, design, programming, and testing—sits together in a open workspace. When you have a question, you need only turn your head and ask. You get an instant response, and if something isn’t clear, you can discuss it at the whiteboard.

Exploiting Great Communication

- Sitting together eliminates the waste caused by waiting for an answer, which dramatically improves productivity.
- The teams delivered software in one-third their normal time.

How can sitting together yield such impressive results? Communication.

Although programming is the emblematic activity of software development, communication is the real key to software success.

Programmers on XP teams spend a far greater percentage of their time programming which increases communication effectiveness of sitting together. Rather than sitting in hour-long meetings, conversations last only as long as needed and involve only the people necessary.

Teams that sit together not only get rapid answers to their questions, they experience what calls Osmotic communication which and when team members are comfortable speaking up like this, it happens often (at least once per day) and saves time and money every time.

Secrets of Sitting Together

- To get the most out of sitting together, be sure you have a complete team.
- It’s important that people be physically present to answer questions. If someone must be absent often—product managers tend to fall into this category—make sure that someone else on the team can answer the same questions.
- Sit close enough to each other that you can have a quick discussion without getting up from your desk or shouting. This will also help encourage osmotic communication, which relies on team members overhearing conversations.
- Available instant help doesn’t do any good if you don’t ask for it. To support this attitude, many teams have a rule: “We must always help when asked.”

Making Room

“Start arranging for a shared workspace now.”

Sitting together is one of those things that’s easy to say and hard to do. It’s not that the act itself is difficult—the real problem is finding space.

Designing Your Workspace

“Leave room for individuality in your workspace.”

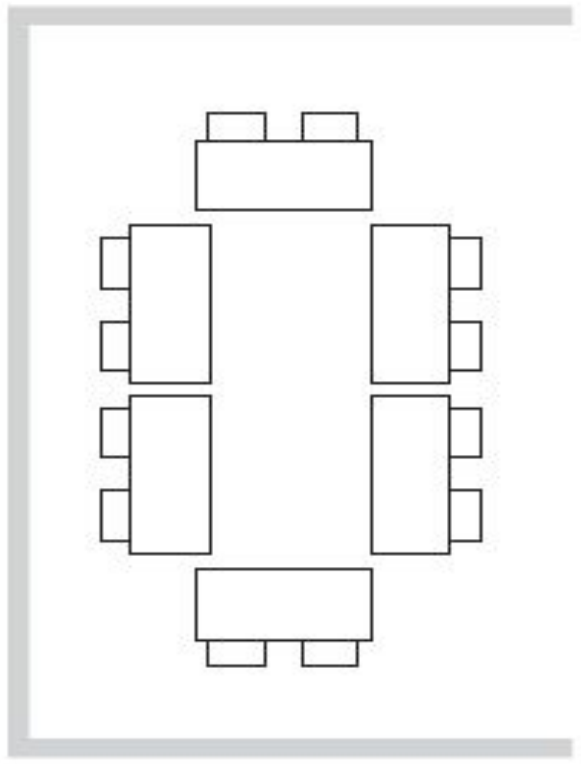
The loss of individuality can make people uncomfortable. Be sure that everyone has a space they can call their own. You also need an additional enclosed room with a door, or cubes away from the open workspace, so people can have privacy for personal phone calls and individual meetings.

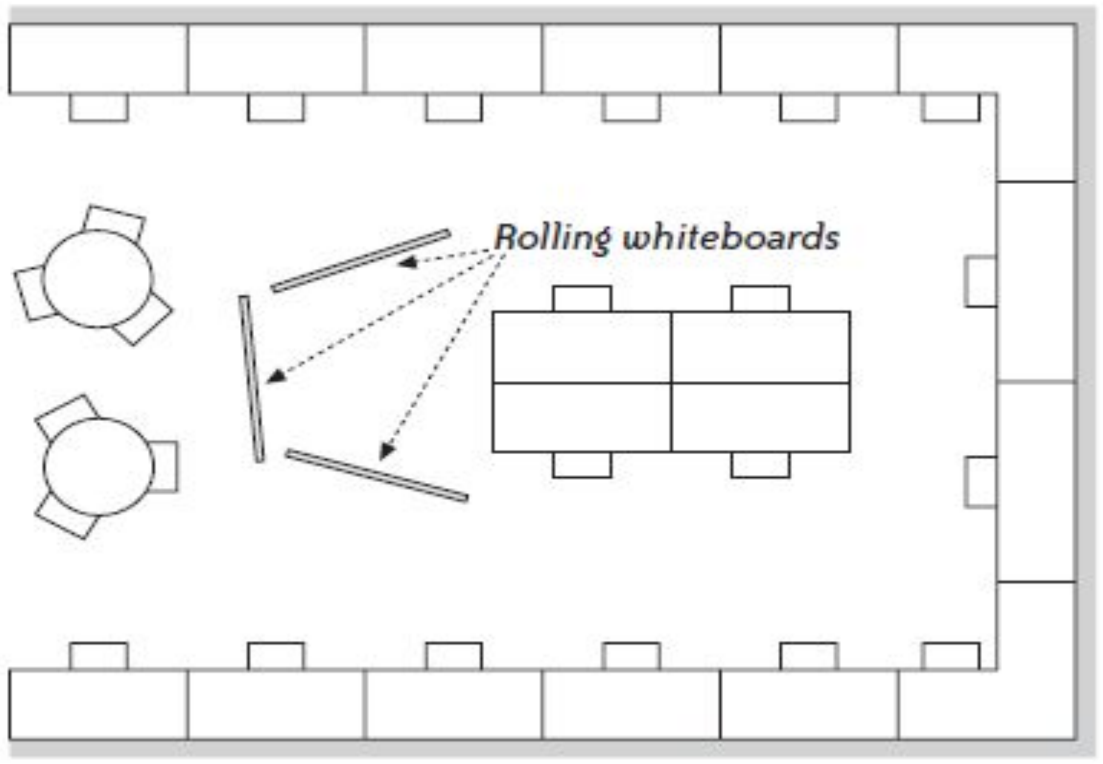
Design your workspace, be sure to include plenty of whiteboards and wall space for an informative workspace.

Some teams include a projector in their workspace. This is a great idea, as it allows the team to collaborate on a problem without moving to a conference room.

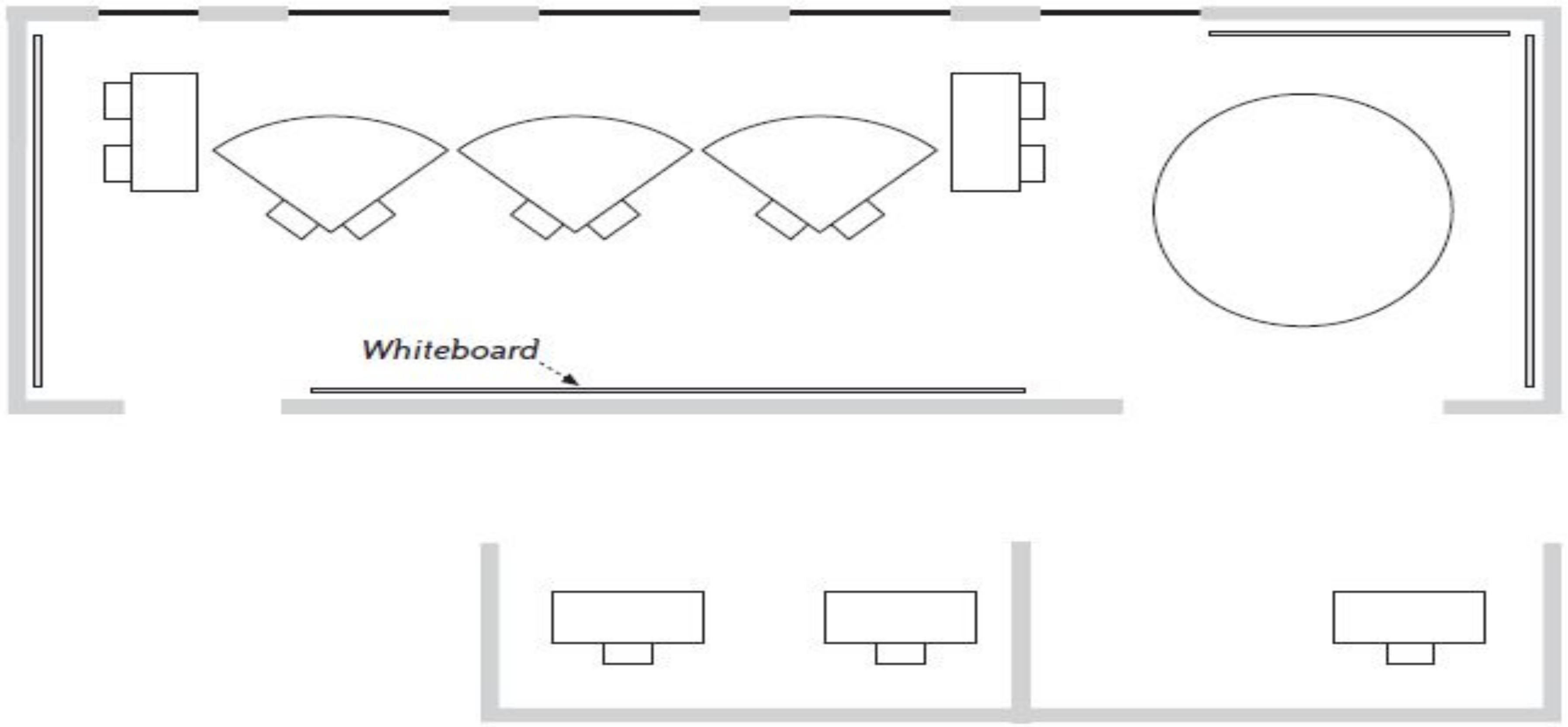
Finally, the center of an XP workspace is typically a set of pairing stations to have the stations facing each other so people can see each other easily.

Sample Workspace





A Small Workspace



Adopting an Open Workspace

“Don’t force people to sit together against their will.”

Some team members may resist moving to an open workspace. Common concerns include loss of individuality and privacy, implied reduction in status from losing a private office, and managers not recognizing individual contributions. Team members may also mention worries about distractions and noise.

Team members initially preferred cubicles to the open workspace, but by the end of the study, they preferred the open workspace.

Results

When your team sits together, communication is much more effective. You stop guessing at answers and ask more questions. You overhear other people’s conversations and contribute answers you may not expect. Team members spontaneously form cross-functional groups to solve problems. There’s a sense of camaraderie and mutual respect.

Contraindications

The hardest part about sitting together is finding room for the open workspace. Cubicles, even adjacent cubicles, won’t provide the benefits that an open workspace does. Start working on this problem now as it can take months to resolve.

Don’t force the team to sit together against their will. Adamant objectors will find a way to leave the team, and possibly the company.

Be careful about sitting together if programmers don’t pair program. Solitary programming requires a quiet workspace. Pair programming, on the other hand, enables programmers to ignore the noise.

Alternatives

If your team is in a single location, you’re better off figuring out how to sit together.

If you have a multisite team, consider turning each site into its own team. For example, if programmers are in one site and customers are in another, the programmers may engage some business analysts to act as proxy customers. In this scenario, the customers and development team should still work together, face-to-face, for several weeks at the beginning of each release.

Real Customer Involvement

Audience: Coaches, Customers

“We understand the goals and frustrations of our customers and end users.”

In an XP team, on-site customers are responsible for choosing and prioritizing features. The value of the project is in their hands. This is a big responsibility—as an on-site customer, how do you know which features to choose?

Some of that knowledge comes from your expertise in the problem domain and with previous versions of the software.

To widen your perspective, you need to involve real customers. The best approach to doing so depends on who you’re building your software for.

Personal Development

In personal development, the development team is its own customer. They’re developing the software for their own use. As a result, there’s no need to involve external customers—the team is the real customer

In-House Custom Development

In-house custom development occurs when your organization asks your team to build something for the organization’s own use. This is classic IT development. It may include writing software to streamline operations, automation for the company’s factories, or producing reports for accounting.

In this environment, the team has multiple customers to serve: the executive sponsor who pays for the software and the end-users who use the software. Their goals may not be in alignment. In the worst case, you may have a committee of sponsors and multiple user groups to satisfy.

Despite this challenge, in-house custom development makes it easy to involve real customers because they’re easily accessible. The best approach is to bring your customers onto the team—to turn your real customers into on-site customers.

Outsourced Custom Development

Outsourced custom development is similar to in-house development, but you may not have the connections that an in-house team does. As a result, you may not be able to recruit real customers to act as the team’s on-site customers.

One way to recruit real customers is to move your team to your customer’s offices rather than asking them to join you at yours.

Vertical-Market Software

Unlike custom development, vertical-market software is developed for many organizations. Like custom development, however, it’s built for a particular industry and it’s often customized for each customer.

Because vertical-market software has multiple customers, each with customized needs, you have to be careful about giving real customers too much control over the direction of the product. You could end up making a product that, while fitting your on-site customer’s needs perfectly, alienates your remaining customers.

Horizontal-Market Software

Horizontal-market software is the visible tip of the software development iceberg: software that’s intended to be used across a wide range of industries. The rows of shrink wrapped software boxes at your local electronics store are a good example of horizontal-market software. So are many web sites.

As with vertical-market software, it’s probably better to set limits on the control that real customers have over the direction of horizontal-market software.

Results

When you include real customers, you improve your knowledge about how they use the software in practice. You have a better understanding of their goals and frustrations, and you use that knowledge to revise what you produce. You increase your chances of delivering a truly useful and thus successful product.

Contraindications

One danger of involving real customers is that they won’t necessarily reflect the needs of all your customers. Be careful that they don’t steer you toward creating software that’s only useful for them.

End-users often think in terms of improving their existing way of working, rather than in terms of finding completely new ways of working. This is another reason why end-users should be involved but not in control.

Alternatives

In the absence of real customer involvement, be sure to have a visionary product manager. It’s best if this person understands the domain well, but you can also hire domain experts to join the team.

Still, feedback from real customers is always informative, even if you choose to ignore it. It’s especially useful when you’ve deployed software to them; their reaction to working software gives you valuable information about how likely you are to reach the greatest levels of success.

Ubiquitous Language

Audience: Programmers

“We understand each other.”

Try describing the business logic in your current system to a nonprogrammer domain expert. Are you able to explain how the system works in terms the domain expert understands? Can you avoid programmer jargon, such as the names of design patterns or coding styles? Is your domain expert able to identify potential problems in your business logic?

If not, you need a ubiquitous language.

The Domain Expertise Conundrum

One of the challenges of professional software development is that programmers aren’t necessarily experts in the areas for which they write software.

Hiring programmers with expertise in a particular domain will reduce this problem, but it won’t eliminate it. In addition, given the choice between a great programmer with no domain experience and a poor programmer with lots of domain experience, One would choose the better programmer.

Two Languages

Programmers program in the language of technology: classes, methods, algorithms, and databases. Domain experts talk in the language of their domain: financial models, chip fabrication plants, and the like.

You could try to translate between the two languages, but it will add delays and errors. You’d produce some software eventually, but you’d probably introduce some bugs along the way. Instead, pick just one language for the whole team to use—a ubiquitous language.

How to Speak the Same Language

- Programmers should speak the language of their domain experts, not the other way around.
- Focus on domain terms rather than technical terms.

Ubiquitous Language in Code

As a programmer, you might have trouble speaking the language of your domain experts. When you're working on a tough problem, it's difficult to make the mental translation from the language of code to the language of the domain.

A better approach is to design your code to use the language of the domain. You can name your classes, methods, and variables anything. Why not use the terms that your domain experts use?

Refining the Ubiquitous Language

The ubiquitous language informs programmers, but the programmers’ need for rigorous formalization also informs the rest of the team.

- First, ensure that the whole team—especially the domain experts—understands and agrees with the changes you’re proposing.
- Second, check that the changes clarify your understanding of the business requirements.
- Third, update the design of the software with the change. The model and the ubiquitous language must always stay in sync. A change to the language implies a change to the model.

Results

When you share a common language between customers and programmers, you reduce the risk of miscommunication. When you use this common language within the design and implementation of the software, you produce code that’s easier to understand and modify.

When the whole team uses the ubiquitous language while sitting together, everyone can overhear domain discussions, especially during pairing sessions. Team members overhear domain and implementation discussions and join in the conversation to resolve questions and expose hidden assumptions.

Contraindications

If you don’t have any domain experts sitting with your team, you may have trouble understanding the domain experts’ thought process deeply enough to have a ubiquitous language. Attempting a ubiquitous language is even more important in this situation, though, as it will allow you to communicate more effectively with domain experts when you do have the opportunity to speak with them.

Alternatives

It’s always a good idea to speak the language of your domain experts. However, avoiding a domain centric design can lead to simpler designs in small, technology-centric projects involving trivial business rules. Be careful, though: this design approach leads to defects and complex, unmaintainable designs in larger projects.

Stand-Up Meetings

Audience: Whole Team

“We know what our teammates are doing.”

A manager reads a list of tasks and asks about each one in turn.

There’s a good reason that organizations hold status meetings: people need to know what’s going on. XP projects have a more effective mechanism: informative workspaces and the daily stand-up meeting.

How to Hold a Daily Stand-Up Meeting

A stand-up meeting is very simple. At a pre-set time every day, the whole team stands in a circle. One at a time, each person briefly describes new information that the team should know.

Some teams use a formal variant of the stand-up called the Daily Scrum. It comes from an agile process also called Scrum. In the Daily Scrum, participants specifically answer three questions:

1. What did I do yesterday?
2. What will I do today?
3. What problems are preventing me from making progress?

“Don’t wait for the stand-up to start your day.”

One problem with stand-up meetings is that they interrupt the day. This is a particular problem for morning stand-ups; because team members know the meeting will interrupt their work, they sometimes wait for the stand-up to end before starting to work. If people arrive at different times, early arrivals sometimes just waste time until the stand-up starts. You can reduce this problem by moving the stand-up to later in the day, such as just before lunch.

Be Brief

The purpose of a stand-up meeting is to give everybody a rough idea of where the team is. It’s not to give a complete inventory of everything happening in the project. The primary virtue of the stand-up meeting is brevity. That’s why we stand: our tired feet remind us to keep the meeting short.

“Thirty seconds per person is usually enough.”

Each person usually only needs to say a few sentences about her status. Thirty seconds per person is usually enough. More detailed discussions should take place in smaller meetings with only the people involved.

Brevity is a tough art to master. To practice, try writing your statement on an index card in advance, then read from the card during the stand-up.

Another approach is to timebox the stand-up. Set a timer for 5 or 10 minutes, depending on the size of the team. When the timer goes off, the meeting is over.

Results

When you conduct daily stand-up meetings, the whole team is aware of issues and challenges that other team members face, and it takes action to remove them. Everyone knows the project’s current status and what the other team members are working on.

Contraindications

Don’t let the daily stand-up stifle communication. Some teams find themselves waiting for the stand-up rather than going over and talking to someone when they need to. If you find this happening, eliminating the stand-up for a little while may actually improve communication.

Alternatives

If you can’t conduct a daily stand-up meeting, you need to stay in touch in some other way. If your team sits together, the resulting natural communication may actually be sufficient. Watch for unpleasant surprises that more communication can prevent.

Another alternative is the traditional weekly status meeting. I find these more effective when team members submit their statuses to a single moderator who can present collated information in 10 or 15 minutes. However, I’ve also seen this approach fall apart quickly.

Coding Standards

Audience: Programmers

“We embrace a joint aesthetic.”

Individual style is great when you’re working alone. In team software development, however, the goal is to create a collective work that is greater than any individual could create on his own. Arguing about whose style is best gets in the way; it’s easier to work together in a single style.

XP suggests creating a coding standard: guidelines to which all developers agree to adhere when programming.

How to Create a Coding Standard

“The most important thing you will learn is how to disagree.”

Creating a coding standard is an exercise in building consensus. It may be one of the first things that programmers do as a team. Over time, you’ll amend and improve the standards. The most important thing you may learn from creating the coding standard is how to disagree constructively.

To that end, I recommend applying two guidelines:

1. Create the minimal set of standards you can live with.

2. Focus on consistency and consensus over perfection.

The best way to start your coding standard is often to select an industry-standard style guide for your language. This will take care of formatting questions and allow you to focus on design-related questions. If you’re not sure what it should encompass, starting points include:

- Development practices
- Tools, key bindings, and IDE
- File and directory layout
- Build conventions
- Error handling and assertions
- Approach to events and logging
- Design conventions

Limit your initial discussion to just one hour. Write down what you agree on. If you disagree about something, move on. You can come back to it later.

Dealing with Disagreement

It’s possible to pressure a dissenter into accepting a coding standard she doesn’t agree with, but it’s probably not a good idea. Doing so is a good way to create resentment and discord.

Instead, remember that few decisions are irrevocable in agile development; mistakes are opportunities to learn and improve.

Go ahead and leave the contested item out of the standard. Maybe lack of standardization in that area will lead to a mess. If it does, the team will learn from the experience and you can change the standard.

Results

When you agree on coding standards and conventions, you improve the maintainability and readability of your code. You can take up different tasks in different subsystems with greater ease. Pair programming moves much more smoothly, and you look for ways to improve the express ability and robustness of your code as you write it.

Contraindications

Don’t allow coding standards to become a divisive issue for your team.

Alternatives

Some teams work together so well that they don’t need a written coding standard; their coding standard is implicit.

If you have a new team, however, create a written coding standard even if everybody gets along well. New teams often go through an initial honeymoon period in which team members are reluctant to disagree with each other. Eventually, disagreements will come out. It’s much better to create a standard before problems escalate.

Adhering to the Standard

People make mistakes. Pair programming helps developers catch mistakes and maintain self-discipline.

Collective code ownership also helps people adhere to the standard, because many different people will edit the same piece of code.

There are less effective approaches. Some teams use automated tools to check their source code for adherence to the coding standard.

“Assume your colleagues are professional and well-meaning.”

Assume your colleagues are professional and well-meaning. If someone is not following the standard, assume that there’s a good reason—even if all the evidence is to the contrary. Your challenge is to find that reason and address it. This approach shows respect for others and will improve others’ respect for you.

Iteration Demo

Audience: Product Manager, Whole Team

“We keep it real.”

An XP team produces working software every week, starting with the very first week.

The iteration demo is a powerful way to do so. First, it’s a concrete demonstration of the team’s progress. The team is proud to show off its work, and stakeholders are happy to see progress.

“Iteration demos help keep the team honest.”

Finally, the demo is an opportunity to solicit regular feedback from the customers. Nothing speaks more clearly to stakeholders than working, usable software.

Regular delivery is central to successful XP. The iteration demo is a concrete indication of that progress.

How to Conduct an Iteration Demo

- Anybody on the team can conduct the iteration demo, but I recommend that the product manager do so.
- Invite anybody who’s interested. The whole team, key stakeholders, and the executive sponsor should attend as often as possible. Include real customers when appropriate.
- The entire demo should take about 10 minutes.
- Once everyone is together, briefly describe the features scheduled for the iteration and their value to the project. If the plan changed during the middle of the iteration, explain what happened.
- After your introduction, go through the list of stories one at a time. Read the story, add any necessary explanation, and demonstrate that the story is finished. Use customer tests to demonstrate stories without a user interface.
- Once the demo is complete, tell stakeholders how they can run the software themselves.

Two Key Questions

At the end of the demo, ask your executive sponsor two key questions:*

1. Is our work to date satisfactory?

2. May we continue?

These questions help keep the project on track and remind your sponsor to speak up if she’s unhappy.

Take immediate action to correct the problem.

Weekly Deployment Is Essential

The iteration demo is a way to prove that you’re making real progress every iteration. Always provide an actual release that stakeholders can try for themselves after the demo.

Even if they are not interested in trying a demo release, create it anyway; with a good automated build, it takes only a moment. If you can’t create a release, your project may be in trouble.

Results

When you conduct a weekly iteration demo and demo release, you instill trust in stakeholders, and the team is confident in its ability to deliver. You share problems forthrightly, which allows you to manage them and helps prevent them from ballooning out of control.

Contraindications

Because the iteration demo is highly visible, you may be tempted to fake a demo. You might show a user interface that doesn’t have any logic behind it, or purposefully avoid showing an action that has a significant defect.

Instead, be clear about the software’s limitations and what you intend to do about them. Faking progress leads stakeholders to believe that you have greater capacity than you actually do.

Alternatives

The iteration demo is a clear indication of your ability to deliver: either you have the ability to demonstrate new features every week, or you don’t. Your executive sponsor either gives you permission to continue, or he doesn’t. I’m not aware of any alternatives that provide such valuable feedback.

Some teams conduct real releases at the end of each iteration rather than a demo. This is a great addition to the practice, if you can do it.

Reporting

Audience: Coaches, Upper Management

“We inspire trust in the team’s decisions.”

Everybody sits together. An informative workspace clearly tracks your progress. All the information you need is at your fingertips.

Why do you need reports?

Actually, you don’t need them. The people who aren’t on your team, particularly upper management and stakeholders, do. They have a big investment in you and the project, and they want to know how well it’s working.

Types of Reports

Progress reports are exactly that: reports on the progress of the team, such as an iteration demo or a release plan.

Management reports are for upper management. They provide high-level information that allows management to analyze trends and set goals.

Progress Reports to Provide

XP teams have a pronounced advantage when it comes to reporting progress: they make observable progress every week, which removes the need for guesswork. Furthermore, XP teams create several progress reports as a normal byproduct of their work.

- Vision statement
- Weekly demo
- Release and iteration plans
- Burn-up chart

Progress Reports to Consider

If your stakeholders want more information, consider providing one or more of the following reports. Avoid providing them by default; each takes time that you could spend on development instead.

- Roadmap
- Status email

Management Reports to Consider

Whereas progress reports demonstrate that the team will meet its goals, management reports demonstrate that the team is working well. As with progress reports, report only what you must.

- Productivity
- Throughput
- Defects
- Time usage

Reports to Avoid

- Source lines of code (SLOC) and function points
- Number of stories
- Velocity
- Code quality

Results

Appropriate reporting will help stakeholders trust that your team is doing good work. Over time, the need for reports will decrease, and you will be able to report less information less frequently.

Contraindications

Time spent on reports is time not spent developing. Technically speaking, reports are wasteful because they don’t contribute to development progress. As a result, I prefer to produce as few reports as possible.

To avoid creating reports manually, I use the iteration demo, planning boards, and the burn-up chart as my only reports whenever I can. They are a normal part of the process and require no extra effort to produce. I use webcams or a digital camera to broadcast the boards if necessary.

Alternatives

Frequent communication can sometimes take the place of formal reporting. If this option is available to you, it’s a better option.