COLLABORATING

Communication in the real world is a lot messier than it is in my image. Instead, people have to work together. They have to ask questions, discuss ideas, and even disagree.
Eight practices to help your team and its stakeholders collaborate efficiently and effectively:

- Trust
- Sitting together
- Real customer involvement
- A ubiquitous language
- Stand-up meetings
- Coding standards
- Iteration demos
- Reporting

Trust

Work together effectively and without fear. The team must take joint responsibility for their work. Team members need to think of the rest of the team as "us," not them." Team members must rely on each other for help. When one member of a team encounters a question that she cannot answer, she doesn't hesitate to ask someone who does know the answer. Sometimes these quick questions turn into longer pairing sessions.
Trust is essential for the team to perform this well. You need to trust that taking time to help others won't make you look unproductive. You need to trust that you'll be treated with respect when you ask for help or disagree with someone.
The organization needs to trust the team, too. Trust doesn't magically appear.

Strategies for generating trust in your XP team:
Team Strategy 1 : Customer-Programmer Empathy

Customers often feel that programmers don't care enough about their needs and deadlines, some of which, if missed, could cost them their jobs. Programmers often feel forced into commitments they can't meet, hurting their health and relationships.

Sometimes, programmers react by inflating estimates and focusing on technical toys at the expense of necessary features; customers react by ignoring programmer estimates and applying schedule pressure.

The biggest missing component in this situation is empathy for the other group’s position. Programmers has to remember that customers have corporate masters that demand results. Bonuses, career advancement, and even job security depend on successful delivery. In the same manner Customers has to remember that ignoring or overriding programmers professional recommendations about timelines often leads to serious personal consequences for programmers.

Sitting together is the most effective way I know to build empathy. Each group gets to see that the others are working just as hard. Retrospectives also help.

Team Strategy 2: Programmer-Tester Empathy

Programmers tend not to show respect for the testers abilities, and testers see their mission as shooting down the programmers work.

Programmers has to remember that testing takes skill and careful work, just as programming does. Take advantage of testers abilities to find mistakes you would never consider, and thank them for helping prevent embarrassing problems from reaching stakeholders and users.

Testers has to focus on the teams joint goal: releasing a great product. When tester finds a mistake, it’s not an occasion for celebration or gloating. Tester has to remember that everybody makes mistakes, and mistakes aren’t a sign of incompetence or laziness.

Team Strategy 3: Eat Together

Another good way to improve team cohesiveness is to eat together. If meal is brought into the office, set a table and serve the food family-style to prevent people from taking the food back to their desks. When go to a restaurant, ask for a single long table rather than separate tables.

Team Strategy 4: Team Continuity

After a project ends, the team typically breaks up. All the wonderful trust and cohesiveness that the team has formed is lost. The next project starts with a brand-new team, and they have to struggle through the four phases of team formation all over again.

This can be avoided by keeping productive teams together. Rather than assigning people to projects, assign a team to a project. Have people join teams
and stick together for multiple projects.

Organizational Strategies for generating trust
Organizational Strategy 1: Show some hustle:
It’s the sense that the team is putting in a fair day’s work for a fair day’s pay. Energized work, an informative workspace, appropriate reporting, and iteration demos all help convey this feeling of productivity.

Organizational Strategy 2: Deliver on Commitments:
Stakeholders probably don’t know much about software development. That puts them in the uncomfortable position of relying on your work, having had poor results before, and being unable to tell if your work is any better.
Meanwhile, team consumes thousands of dollars every week in salary and support. How do stakeholders know that team is spending their money wisely? How do they know that the team is even competent?
Stakeholders may not know how to evaluate the process, but they can evaluate results. Two kinds of results speak particularly clearly to them: **working software and delivering on commitments**.
Fortunately, XP teams demonstrate both of these results every release. Team
makes a commitment to deliver working software in iteration and release plans. And the same is demonstrated in the iteration demo. All the thing team has to do is create a plan that you can achieve... and then achieve it.

Organizational Strategy 3: Manage Problems

When expected problem arise, work on the hardest, most uncertain tasks early in the iteration.

Start by letting the whole team know about it. Bring it up by the next stand-up meeting which gives the entire team a chance to help solve the problem.

If the problem is relatively small, it can be absorbed into the iteration by using some of your iteration slack. Options for absorbing the problem include reducing noncritical refactorings, postponing a nonessential meeting, or even cancelling research time.

Some problems are too big to absorb no matter how much slack you have. If this is the case, get together as a whole team as soon as possible and replan. You may need to remove an entire story or you might be able to reduce the scope of some stories.

When you’ve identified a problem, let the stakeholders know about it. They’ ll appreciate your professionalism even if they don’t like the problem. The sooner your stakeholders know about a problem the more time they have to work around it. Include an analysis of the possible solutions as well as their technical costs. It can take a lot of courage to have this discussion– but addressing a problem successfully can build trust . When a problem does occur, you should usually be able to solve it by using slack, not overtime.

Organizational Strategy 4: Respect Customer Goals

When starting a new XP project, programmers should make an extra effort to welcome the customers. One particularly effective way to do so is to treat customer goals with respect. Being respectful goes both ways, and customers should also suppress their natural tendencies to complain about schedules and argue with estimates.

Another way for programmers to take customer goals seriously is to come up with creative alternatives for meeting those goals. As programmers and customers have communicate each other barriers will be broken and trust will develop.

Organizational Strategy 4: Promote the Team
Promote the team directly: One team posted pictures and charts on the outer wall of the workspace that showed what they were working on and how it was progressing. Another invited anyone and everyone in the company to attend its iteration demos.

Being open about what you’re doing will also help people appreciate your team.

Organizational Strategy 6: Be Honest

Borderline behavior includes glossing over known defects in an iteration demo, taking credit for stories that are not 100 percent complete, and extending the iteration for a few days in order to finish everything in the plan.

All of these behaviors give stakeholders the impression that you’ve done more than you actually have. There’s a practical reason not to do these things: stakeholders will expect you to complete the remaining features just as quickly. Backlog can be updated that looks done but isn’t. At some point, you’ll have to finish that backlog, and the resulting schedule slip will produce confusion, disappointment, and even anger.

In a desire to look good, teams sometimes sign up for more stories than they can implement well. They get the work done, but they take shortcuts and don’t do enough

design and refactoring. It results in more defects and the team finds itself suddenly going much slower while struggling to improve code quality.
Similarly, don’t yield to the temptation to count stories that aren’t “done done”. If a story isn’t completely finished, it doesn’t count toward your velocity. Don’t even take partial credit for the story.

<u>Ubiquitous Language</u>

Ubiquitous language helps to understand each other.

The Domain Expert Conundrum

The people who are experts in the problem domain—the domain experts— are rarely qualified to write software. The people who are qualified to write software—the programmers—don’t always understand the problem domain.

Domain experts communicate their expertise to programmers, who in turn encode that knowledge in software. The challenge is communicating that information clearly and accurately.

Ex: If a health care domain expert speaks healthcare terminology, programmers might not understand. If programmers speaks in terms of API names, classes or methods domain expert will not understand.

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, healthcare terms.
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?

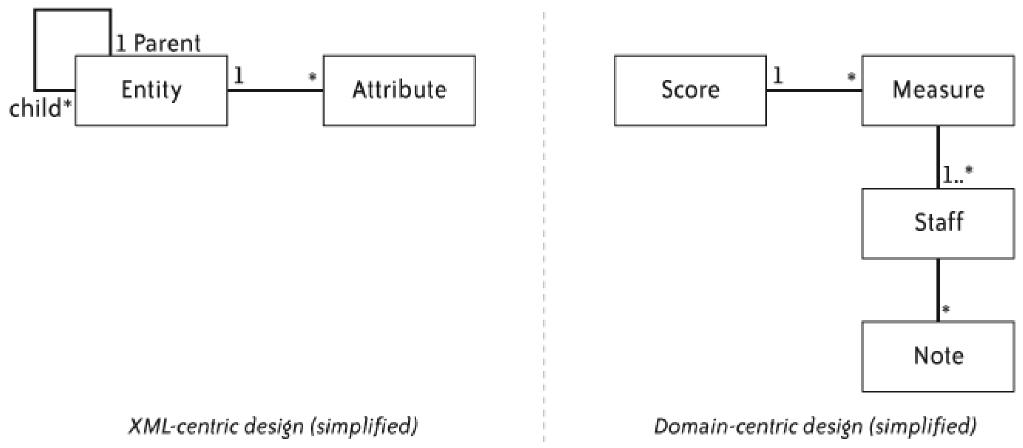
Programmers should speak the language of their domain experts, not the other way around.

If programmer asks question including domain terms, domain expert understands the question and clarifies it.

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. Use the domain terms for classes, methods, variables.
One powerful way to design your application to speak the language of the domain is to create a domain model.

Ex:


1 Parent
child*
Entity
1
*
Attribute
XML-centric design (simplified)
Score
1
*
Measure
1..*
Staff
*
Note
Domain-centric design (simplified)

Refining the Ubiquitous Language

The ubiquitous language guides programmers. Often, programmers' questions lead domain experts to rethink their assumptions. Therefore, ubiquitous language is a living language it must reflect reality. As understanding improves, the language should evolve too, with a few important cautions.

- ensure that the whole team especially the domain experts understands and agrees with the changes you're proposing.
- check that the changes clarify your understanding of the business requirements. It may seem clearer to make a change, but the language must still reflect what the users need to accomplish with the software.

- update the design of the software with the change. The model and the ubiquitous language must always stay in sync. Yes, this does mean that you should refactor the code when your understanding of the domain changes.

Stand-up meetings

Organizations hold status meetings 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

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.

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?

One problem with stand-up meetings is that they interrupt the day. Don't wait for the stand-up to start your day.

Be Brief

The purpose of a stand-up meeting is to give everybody a rough idea of where the team is. 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.
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.

Ex: Programmer – Yesterday I have completed code changes and testing for story1, today I will do analysis of story2.

Coding Standards

Each programmer has his/her own way of producing code.
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.
XP suggests creating a coding standard: guidelines to which all developers agree to adhere when programming.

Beyond Formatting

Uniform coding formatting is important while coding as team. Otherwise, the layout may look ugly.
Ex: One programmer uses 2 spaces for tab, another programmer uses 4 spaces for tab.

Coding standards are more than formatting.

- clearly named variables and methods
- use assertions to make code fail fast
- never null references between objects
- never include not reachable code
- avoid unused imports
- handling exceptions
- when and where to log events

How to create a coding standard
Creating a coding standard is an exercise in building an agreement. The most important thing you may learn from creating the coding standard is how to disagree constructively.
Apply two guidelines:
1. Create the minimal set of standards you can live with.
2. Focus on consistency and consensus over perfection.

Discuss coding standard in the **first iteration**.After initial talks on vision, stories, and planning, customers and testers focus on the release plan—this is the ideal time for programmers to agree on coding standards.

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.

Starting points to create coding standards include:

- Development practices
- Tools, keybindings, and IDE
- File and directory layout
- Build conventions
- Error handling and assertions
- Approach to events and logging
- Design conventions (such as how to deal with null references)

Write down what you agree on. If you disagree about something, move on. You can come back to it later.

If discussions stall, refocus on the **software goals and desired outcomes** and agree on those first. Make key decisions, move forward, and improve the standards over time.

If discussions become heated, use a **facilitator** to keep the focus on team goals. Hold short follow-up meetings after a few days and a few weeks to review and adjust standards based on experience. If disagreements remain, try one approach, then revisit it later.

Update coding standards whenever needed with team agreement and use retrospectives to discuss changes.

Dealing with disagreement

Pressuring someone to accept a coding standard they disagree with can create resentment and harm teamwork. In agile development, most decisions are **not permanent** mistakes can be corrected and used as learning opportunities.

As Ward Cunningham explains, you don't need to win every argument. If there is disagreement, try one approach and see what happens. If it causes problems, the team can change it later.

So, if a rule is strongly contested, leave it out for now.

Adhering to the Standard

Pair programming helps developers catch mistakes and maintain self-discipline. It provides a way to discuss formatting and coding questions not addressed by the guidelines. It's an also an excellent way to improve the standard; it's much easier to suggest an improvement when you can talk it over with someone first.

Collective code ownership also helps people adhere to the standard, because many different people will edit the same piece of code.
Some teams use automated tools to check their source code for adherence to the coding standard. Others program their version control system to reformat files upon check-in.

If someone is not following the standard, find the reason and address it. This approach shows respect for others and will improve others respect for you. Start by talking to colleague alone about the standard. If the objector does not agree, discuss with rest of the team and consider changing the standard. If the objector agrees but is not applying, then ask again in polite way. During discussion, you may learn that the objector did not understand the standard. Then explain the standard. If objector is a junior programmer and needs help, coordinate with the rest of the team for pair programming.

Iteration Demo

An XP team produces working software every week, starting with the very first week. It takes a lot of discipline to keep that pace.

Programmers need discipline to keep the code clean so they can continue to make progress. Customers need discipline to fully understand and communicate one set of features before starting another. Testers need discipline to work on software that changes daily.

The rewards for this hard work are significantly reduced risk, a lot of energy and fun, and the satisfaction of doing great work and seeing progress. The biggest challenge is keeping your momentum.

The iteration demo is a powerful way to do so. First, it's a concrete demonstration of the team's progress. Second, the demos help the team be honest about its progress.

Iteration demos are open to all stakeholders, and some companies even invite external customers to attend.

Nothing speaks more clearly to stakeholders than working, usable software. Demonstrating your project makes it and your progress immediately visible and concrete. It gives stakeholders an opportunity to understand what they're getting and to change direction if they need to.

How to conduct an Iteration Demo

- Anybody on the team can conduct the iteration demo, but product manager is preferred. He has the best understanding of the stakeholder's point of view and speaks their language. His leadership also emphasizes the role of the product manager in steering the product.
- The whole team, key stakeholders, and the executive sponsor should attend and interested ones are invited.

- Include real customers when required.
- Use a teleconference and desktop-sharing software if no one available in a room.
- The entire demo should take about 10 minutes.
- Both the product manager and the demo should be available for further discussion and exploration after the meeting.
- 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. Full disclosure will raise your credibility.
- After 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.
- Once the demo is complete, tell stakeholders how they can run the software themselves.
- Make an installer available on the network, or provide a server for stakeholder use.

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.

- If answer is “no” or “yes” very reluctantly for the first question,
  - Indicates that something is wrong
  - Talk with sponsor after demo
  - Take immediate action
- If continuation is “no” for second question (rare):
  - Confirm the decision with the sponsor
  - Offer to ship the current work and request a final wrap-up week
  - Understand what went wrong and include the sponsor in the retrospective

Weekly Deployment is Essential
Iteration demos are not just for show, they prove real progress each iteration. Teams should always create a working release that stakeholders can try. If a release can't be created, it signals project risk. Regular demo releases reduce the gap between "done" and "shipped," lower schedule risk, and help control technical debt. New projects should ensure code is deployable every iteration, while legacy projects that can't do this yet reveal technical debt that must be addressed.

Reporting

Why do you need reports?
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

1. Progress reports are exactly that: reports on the progress of the team, such as an iteration demo or a release plan. Although progress reports seem to exist so that stakeholders can monitor and correct the team's direction, that's not their purpose. Instead, good progress reports allow stakeholders to trust the team's decisions.

2. Management reports are meant for upper management and provide high-level data, such as trends in productivity and defects, to help analyze performance and set goals.

What kinds of reports do you need to build trust and satisfy strategic needs?
It depends on the stakeholders. Some stakeholders are hands-off and just want to see progress toward a goal; others want to know more details so they can build broader knowledge. Provide just enough reporting to satisfy key stakeholders.

Progress reports to provide
XP teams make observable progress every week, which removes the need for guesswork.

Vision Statement – On-site customers create and update a vision statement that describes what you're doing, why you're doing it, and how you'll know if you're successful

Weekly Demo - Nothing is as powerful at demonstrating progress as working software. Invite stakeholders to the weekly iteration demo.

Release and iteration plans - The release and iteration planning boards already posted in your workspace provide great detail about progress. Invite stakeholders to look at them any time they want detailed status information.

Burn-up chart – It shows progress and predicts a completion date. Most teams produce a burn-up chart when they update their release plan.

Progress reports to consider
If your stakeholders want more information, consider providing one or more of the following reports

Roadmap - Some stakeholders may want more detail than the vision statement provides. For these stakeholders, consider maintaining a document or slide deck that summarizes planned releases and the significant features in each one.

Status Email - A weekly status email can supplement the iteration demo. Include list of the stories completed for each iteration and their value. Include our current range of probable completion scope and dates. If you added or removed stories from the schedule, explain that here.

Management Reports to Consider
Whereas progress reports demonstrate that the team will meet its goals, management reports demonstrate that the team is working well

Productivity – Productivity is the amount of production over time. Instead of trying to measure features, measure the team's impact on the business. Create an objective measure of value, such as return on investment. Product manager and upper management should be able to help create this measure. Once you have a measure, track its value every iteration. Until the team releases software to production, this number will trend downward, below zero. The team will be incurring costs but not generating value. After a release, the trend should turn upward. For your team to achieve an organizational success, not just a technical success, your software must provide business value. This productivity metric reflects that fact.

Throughput - Throughput is the number of features the team can develop in a particular amount of time. To avoid difficult questions such as “What's a feature?

” measure the amount of time between the moment the team agrees to develop some idea and the moment that idea is in production and available for general use. The less time, the better.

Defects-On an XP team, finding and fixing defects is a normal part of the process. To avoid overcounting defects, wait until you have marked a story as “done done” and completed its iteration before marking something as a defect.

Time Usage - If the project is under time pressure—and projects usually are—stakeholders may want to know that the team is using its time wisely.
produce a report that shows how the programmers
are using their time. This report requires that programmers track their time in detail.

Reports to avoid

Source Lines of Code(SLOC) and function points - Source lines of code (SLOC) and its language-independent cousin, function points, are common approaches to measuring software size. Unfortunately, they’re also used for measuring productivity.Well-designed code is modular; it supports multiple features without duplication. The better the design, the less duplication, and thus the fewer lines of code. This sort of careful design takes time and effort, but results in fewer bugs and software that’s easier to change.

Number of stories - Some people think they can use the number of stories delivered each iteration as a measure of productivity. Stories have nothing to do with productivity.

Velocity - Different teams will have different ways of estimating. Their velocities have nothing in common.

Code Quality – There’s no substitute for developer expertise in the area of code quality. The available code quality metrics, such as cyclomatic code complexity, all require expert interpretation. There is no single set of metrics that clearly shows design or code quality. The metrics merely recommend areas that deserve further investigation. Avoid reporting code quality metrics. They are a useful tool for developers